Towards AI-assisted Healthcare: System Design and Deployment for Machine Learning based Clinical Decision Support

by

Andong Zhan

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

March, 2018

Abstract

Over the last decade, American hospitals have adopted electronic health records (EHRs) widely. In the next decade, incorporating EHRs with clinical decision support (CDS) together into the process of medicine has the potential to change the way medicine has been practiced and advance the quality of patient care. It is a unique opportunity for machine learning (ML), with its ability to process massive datasets beyond the scope of human capability, to provide new clinical insights that aid physicians in planning and delivering care, ultimately leading to better outcomes, lower costs of care, and increased patient satisfaction. However, applying ML-based CDS has to face steep system and application challenges. No open platform is there to support ML and domain experts to develop, deploy, and monitor ML-based CDS; and no end-to-end solution is available for machine learning algorithms to consume heterogenous EHRs and deliver CDS in real-time. Build ML-based CDS from scratch can be expensive and time-consuming.

In this dissertation, CDS-Stack, an open cloud-based platform, is introduced to help ML practitioners to deploy ML-based CDS into healthcare practice. The CDS-

ABSTRACT

Stack integrates various components into the infrastructure for the development, deployment, and monitoring of the ML-based CDS. It provides an ETL engine to transform heterogenous EHRs, either historical or online, into a common data model (CDM) in parallel so that ML algorithms can directly consume health data for training or prediction. It introduces both pull and push-based online CDS pipelines to deliver CDS in real-time. The CDS-Stack has been adopted by Johns Hopkins Medical Institute (JHMI) to deliver a sepsis early warning score since November 2017 and begins to show promising results. Furthermore, we believe CDS-Stack can be extended to outpatients too. A case study of outpatient CDS has been conducted which utilizes smartphones and machine learning to quantify the severity of Parkinson disease. In this study, a mobile Parkinson disease severity score (mPDS) is generated using a novel machine learning approach. The results show it can detect response to dopaminergic therapy, correlate strongly with traditional rating scales, and capture intraday symptom fluctuation.

Primary Reader: Dr. Andreas Terzis

Secondary Readers: Dr. Suchi Saria, Dr. Yanif Ahmad, and Dr. David N. Hager

Acknowledgments

My journey began in August 2010, and a lot of people have helped me throughout this long and excellent adventure in my life which recreated myself.

The first person that I would like to thank is my advisor, Dr. Andreas Terzis. Andreas gave me the chance to start my research at Hopkins and always supported and encouraged me to pursue my interests in research. I enjoyed the autonomy he gave to me to explore and take the risk on my Ph.D. study and when I was stuck he always guided and helped me to find a way out. I also would like to thank Dr. Suchi Saria. I feel so lucky to have her to be my co-advisor and led me to explore and solve the real problems in healthcare. She set an example that I can follow to learn how to research an interdisciplinary domain. I also would like to thank Dr. Yanif Ahmad. Yanif and I worked closely together in the last year of my Ph.D., and he continuously shared his broad knowledge and rich experience in the computer system and cloud deployment. I couldn't imagine to build and deploy such a large and complex system without him. I learned a lot from him, and he also helped me to build my confidence in software engineering. I also want to thank other GBO

and thesis committee members, Dr. Thomas Woolf and Dr. David N. Hager. Their precious and insightful feedback has improved my research to reach a level that I could not achieve by myself.

I am honored to work with some excellent lab-mates from two great labs: the Hopkins interNetworking Research Group (HiNRG) and the machine learning and healthcare lab. I would like to thank Dr. Chieh-Jan Mike, Dr. Jong Hyun Lim, Dr. Yin Chen, Dr. JeongGil Ko, Dr. Douglas Carlson, and Dr. Marcus Chang from HiNRG to guided and helped me start my research at Hopkins. They inspired me during every discussion in the group meeting and share their insights unconditionally. Dr. Jong Hyun Lim introduced me to work with him in his mobile sensing project which captured my interest on this domain, and Dr. Marcus Chang was a great mentor who helped me to explore and learn embedded systems. I would like to thank Katie Henry, Hossein Soleimani, Srihari Mohan, San-He Wu, Dr. Nishi Rawat from the machine learning and healthcare lab. They showed me how to use machine learning and statistics to solve real healthcare problems, and I have learned a lot from them on clinical care.

Performing the research into healthcare practice would not have been possible without my colleagues who are doctors and health informatics experts. I would like to thank Dr. Peter Greene, the CMIO at JHMI, who believed our vision and campaigned for us to apply our idea and system into JHMI. I would like to thank Diana Gumas, Bonnie Woods, and Matthew Toerper from the Center for Clinical

ACKNOWLEDGMENTS

Data Analysis (CCDA) to help us access the real dataset from JHMI. I would like to thank Carla Sproge, Lisa Moore, Abby Podratz, and other engineers from IT at JHMI to work with us to integrate the system with Epic EHR system at JHMI. Also, I would like to thank the team from Department of Neurology, University of Rochester Medical Center, to conduct the remote monitoring and clinical assessment for Parkinson patients. I especially want to thank Dr. Ray Dorsey, Dr. Christopher Tarolli, and Dr. Max Little for their cooperation and leadership on the Parkinson project.

Last but not the least, I am deeply grateful to my family. I want to thank my parents for their unconditional love and support. My parents have always been my cheerleaders even though they cannot watch the game on-site. I thank my wife Kelly for her boundless and unfading love. She always believes in me and encourages me never giving up. She defines the word "love" in my vocabulary. Without her accompany, I wouldn't have gotten through the long journey of Ph.D. I also thank my parents-in-law for their love and support.

Dedication

To my parents for all their unconditional love and support. I appreciate they are always thinking of me on the other side of the earth.

To Kelly, my wife, for her unending support, love, and belief. I wouldn't have gotten through this doctorate if it wasn't for her.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> *"Human beings, in all lines of work, make errors. Errors can be prevented by designing systems that make it hard for people to do the wrong thing and easy for people to do the right thing."*
>
> – Kohn et al., *To Err Is Human: Building a Safer Health System*[1]

Less than a decade ago, nine out of ten doctors in the U.S. updated their patients' records by hand and stored them in color-coded files. By the end of 2017, approximately 90% of office-based physicians nationwide would be using electronic health records (EHRs).[2] The potential benefits of EHRs over traditional paper are many, including cost containment, reductions in errors, and improved compliance by utilizing real-time data. The highest functional level of the EHR is clinical decision support (CDS) and process automation, which are expected to enhance patient health and healthcare. [3] Therefore, in the next decade, incorporating EHRs with clinical

decision support together into the process of medicine has the potential to change the way medicine has been practiced and advance the quality of patient care.[4]

1.1 Machine Learning-based Clinical Decision Support System

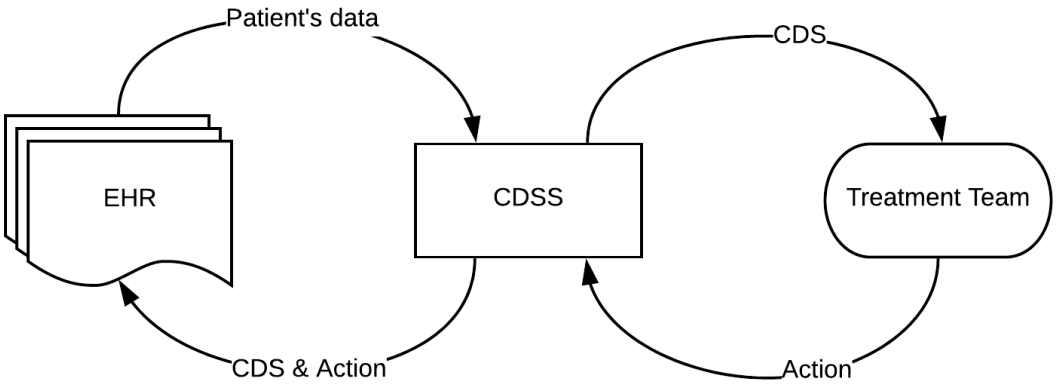


Figure 1.1: Clinical Decision Support System (CDSS).

A clinical decision support system (CDSS), as shown in Fig. 1.1, is a system to request the patients' data and in response, proposes a set of appropriate clinical decision support, e.g., diagnoses and treatments. The doctor then takes the output and determines which diagnoses might be relevant and which are not, and if necessary orders further clinical tests to narrow down the diagnosis or confirm the diagnosis with treatment orders. To Err is Human, CDSS has been a key element of systems' approaches to improving patient safety and the quality of care and has been an

essential requirement for "meaningful use" of EHRs.[4]

A CDSS can be either knowledge-based or non-knowledge-based. Knowledge-based CDSSs contain the predefined rules (e.g., health-related criteria, standards, and guidelines) and associate with patients' data to make the diagnosis. For example, clinically, the Systemic Inflammatory Response Syndrome (SIRS) is identified by two or more symptoms including fever or hypothermia, tachycardia, tachypnoea and, change in blood leucocyte count. This criterion can be easily defined in computer systems (See Table 1.1). However, not all disease diagnosis can be defined as simple and perfect rules. For example, acute kidney injury (AKI) is currently detected by applying a formula (called the AKI algorithm[5]) to NHS patients' blood tests, and severe sepsis and septic shock are diagnosed by using complex inclusion criteria (e.g., Press Ganey's sepsis workflow) in JHMI. These algorithms are good but not perfect: they have a tendency to generate false positives for patients and insensitive to personal clinical information, e.g., patients' age, gender, medical histories, and so on, — all of which makes a difference.

SIRS Definition
Temp $> 38°C$ (100.4°F) or $< 36°C$ (96.8°F)
Heart rate > 90
Respiratory rate > 20 or $PaCO_2 < 32$ mm Hg
WBC $> 12{,}000/mm^3$, $< 4{,}000/mm^3$, or $> 10\%$ bands

Table 1.1: The SIRS criteria (≥ 2 meet SIRS definitions).

Non-knowledge-based CDSSs use a form of artificial intelligence called machine learning (ML). Machine learning allows computers to learn from past experiences and

find patterns in clinical data. Compared with Knowledge-based CDSS, this eliminates the need for writing rules and expert input. Also, the complexity of some diseases, especially for those with an unclear standard for diagnosis in clinical care (e.g., sepsis, rheumatoid arthritis),[6] provides a unique opportunity for machine learning to provide new insights and has stimulated research into novel methods for this purpose. For example, researchers from Johns Hopkins University have shown supervised learning can improve the diagnostic accuracy of sepsis.[7–9] Google has developed a machine learning algorithm to help identify cancerous tumors on mammograms.[10] Machine learning also can help on diagnosis and daily disease management for neurodegenerative diseases like Parkinson's or Alzheimer's disease.[11,12] It's clear that machine learning puts another arrow in the quiver of clinical decision making.

1.2 Challenges to Apply ML-based CDS in Practice

It is still rare for hospitals to adopt machine learning into their healthcare practice, even though recently, more and more promising research studies have been published in using machine learning in healthcare, from diagnosis to prognosis, from inpatients to outpatients, almost covered every corner of healthcare. The ML-based CDS, as an interdisciplinary practice, has to face several steep challenges, across computer systems, machine learning, human-computer interaction, health informatics, patient

safety and quality, ethics, etc. In this dissertation, we address the two fundamental challenges — system and application challenges — in building ML-based CDSS from a computer system perspective.

1.2.1 System challenges

Infrastructure and tooling. Are we ready to deploy a machine learning model published in top journals in machine learning and clinical care? The answer is "NO". It may be surprising to the academic community that the "core" of what academic researchers think of as machine learning — training or prediction — is a comparatively small, albeit critical, part of the overall decision-generation lifecycle. The necessary surrounding infrastructure is vast and complex to support end-to-end applications.[13,14] The infrastructure needs to support the end-to-end data pipelines from extracting EHRs to delivering CDS on the user interface. It should accelerate the process of turning an idea into an online product and make it configurable and reusable. It should introduce tools for us to monitor the incoming EHR data flows, machine learning predictions, CDS outcomes, and user interactions. Also, it should enable applications to be HIPAA secure and compliant.

Integration of various and heterogenous components. We observe that a significant challenge in building ML-based CDSS comes from the heterogeneity of the various components that must be integrated together into production workflows. A production system must run like clockwork, collecting new EHRs in real-time,

splitting out personalized diagnosis every minute, aggregated reports every hour, etc. It is challenging to get a bunch of heterogeneous components to operate as a synchronized and coordinated workflow. Also, since the EHR vendors are both the original data sources and the CDS destinations (i.e., doctor accesses the CDS from the EHR vendor), integrating CDSS with a comparatively closed EHR system is not straightforward.[15]

1.2.2 Application challenges

Understanding clinical workflows is an iterative process. The clinical workflows for some diseases, e.g., AKI and sepsis, are complicated for computer scientists to understand and turn them into programmable rules entirely in one go. From our experience, it is an iterative process to learn from the clinical experts in their practice. Also, some of the documented workflows have not been precisely adopted in practice. Those require the system to be adaptive to the iterative learning process and provides continuous support for your deployment and maintenance.

Prepare EHRs for machine learning is time-consuming. The acquisition, analysis, interpretation, and presentation of EHR data in a clinically relevant and usable format is the premier challenge of data analysis in critical care.[16] First of all, the data acquisition can be slowed down due to limited interoperability. In 2017, there were roughly 1100 vendors that offer an EHR — twice the number of vendors four years ago.[2] Each EHR may provide customized APIs or data schemas for data

access. Even though the on-going standardization of EHR exchange protocols, e.g., FHIR (Fast Healthcare Interoperability Resources), may substantially improve EHR interoperability shortly, the vendor specific APIs are still necessary for the CDSS to access customized clinical events and hospital-specific clinical workflows. Second, feature extraction on EHR requires elaborate work. EHR is heterogeneous and intricate. An EHR database (or enterprise data warehouse, EDW) may contain more than 150,000 tables involving more than 140,000 types of medications, 48,000 of lab results, and 60,000 thousands of timestamped data items. Clinical related data is present in a plethora of formats including lab results, clinical observations, imaging scans, free text notes, genome sequences, continuous waveforms and more. However, most ML systems or algorithms, e.g., Scikit-learn,[17] Spark,[18] and TensorFlow,[19] prefer a data format called "data frame" (a.k.a. tabular data). It is a data structure representing samples as rows, each of which consists of many observations or measurements, i.e., columns; and the data types of the columns are usually numeric (e.g., integer and float numbers). The ETL (extract, transform, load) from EHRs to such a data frame is the first step to fill the data gap between EHR and ML. With a selection of required measurements from the targeted EHR, machine learning also needs some advanced features derived from these measurements. The feature engineering framework needs to maximize reusable code and make it flexible and configurable.

Real-time clinical decision support. Real-time clinical decision support is vital for some CDS applications to deliver interventions at the earliest time, especially

for some acute diseases, e.g., sepsis. Sepsis starts from the source of infections and then spread into the blood and other organs rapidly. Without early warning and intervention, the infection can cause organ dysfunction and even death. Timing is critical for providing CDS and trigger following with treatments. Real-time CDS may not be a technical issue for the CDSS integrated with EHR but could be a challenge if the CDSS is an external application.

Extensibility vs. data dependency. Machine learning models have high data dependency. No input is even genuinely independent for most machine learning models, i.e., Changing Anything Changes Everything (CACE). CACE applies not only to input features, but also to hyper-parameters, learning settings, sampling methods, convergence thresholds, data selection, and essentially every other possible tweak.[13] When deploying an ML algorithm from one site to another, the end-to-end system needs to be rebuilt and re-evaluated again because the incoming EHRs are changed. Specifically, features need to be re-extracted and the ML algorithm needs to be re-trained and verified. Sound system design should minimize the pain in new deployment and accelerate the extension of feasible machine learning-based CDS applications.

1.3 Contributions of the Dissertation

The contributions of this dissertation are as follows. First, *CDS-Stack*, an open cloud-based infrastructure has been provided to catalyze the deployment of machine learning-based CDS. Specifically, 1) the CDS-Stack provides a full stack, across hardware, service, and interface, to support end-to-end machine learning-based clinical decision support systems. The stack spans from data acquisition, feature extraction, machine learning, to CDS integration, security, and monitoring; 2) It provides an ETL engine to transform heterogenous EHRs, either historical or online, into a common data model (CDM) in parallel so that ML algorithms can directly consume health data for training or prediction; 3) It introduces both pull and push-based online CDS pipelines to deliver CDS in real-time.

Second, we have deployed CDS-Stack at three major hospitals in Johns Hopkins Medical Institute (JHMI), and delivered TREWS,[7] the targeted real-time early warning score for sepsis, to Howard County General Hospital (HCGH) since November 2017. The preliminary results showed promising adoption rate by the treatment team in the pilot study.

Third, CDS-Stack can further extend from in-hospital settings to outpatients. A case study was conducted for outpatient CDS. In the case study, a mobile Parkinson disease severity score (mPDS) is generated using a novel machine learning approach. The score 1) detected response to dopaminergic therapy, 2) correlated strongly with traditional rating scales, and 3) captured intraday symptom fluctuation.

1.4 Organization of the Dissertation

The remainder of this thesis is structured as follows. The overview of CDS-Stack and the preliminary results of TREWS deployment is presented in Chapter 2. The CDM and ETL engine are presented in Chapter 3. The pull-based and push-based online ETL pipeline are described in Chapter 4. In Chapter 5, the mPDS, an example of DDSS at home, is presented. Chapter 6 elaborates related work. Finally, Chapter 7 conclude this dissertation.

1.5 Scope of the Dissertation

The primary goal of this dissertation is not to improve machine learning algorithms, which are almost always "good enough" for many critical applications, but instead to make them deployable and actionable so that ML practitioners can quickly apply ML to healthcare practice, achieve high-quality results, and maintain production systems in clinical decision support. In this dissertation, we treat the machine learning algorithms as a black box. Also, the CDS-Stack focuses on using the "main body" of EHR into CDS, which includes demographics, diagnosis, medical history, problem list, medication administration, lab results, lab orders, vital signs, etc. And the current CDS-Stack used the customized API in the targeted EHR, Epic. And we haven't covered the implementation with FHIR since it was still under construction in JHMI. This dissertation does not focus on using specific types of EHRs to imple-

ment CDS, e.g., medical image-related machine learning-based CDS or the NLP of medical notes. This dissertation also focuses on diagnostic decision support, one of the critical categories of CDS. While clinical decision support can be a broader set of tools, including clinical guidelines, condition-specific order sets, focused patient data reports and summaries, documentation templates, etc.

Chapter 2

Overview of the CDS-Stack: Architecture and Deployment

"The workman who would do his work well should first sharpen his tools."

– Confucius, *The Analects of Confucius*

2.1 Introduction

Despite incredible recent advances in machine learning for healthcare, building machine learning-based clinical decision support system remains prohibitively time-consuming and expensive. This expense comes not from a need for new and improved statistical models but instead from a lack of systems and tools for supporting end-to-end machine learning application development, from data preparation and feature

extraction to deployment and monitoring. Therefore, we developed CDS-Stack, an open cloud-based platform for ML practitioners to develop, deploy, and maintain their ML-based CDSS. Using CDS-Stack, developers do not need to build the end-to-end ML-based CDSS from scratch; instead, CDS-Stack provides infrastructure and building blocks for developers to start with. The CDS-Stack is available at `https://github.com/cloud-cds/cds-stack`.

Our design philosophy in the CDS-Stack centers around two central tenets:

- Target end-to-end ML-based CDS workflows. ML-powered CDS development consists of far more than model training. As a result, today, the bulk of challenges in developing new ML-powered CDS are not in model training but are instead in data collection/preparation, feature selection/extraction, and deployment (serving, monitoring, debugging, etc.). Systems should target the entire, end-to-end workflow so that machine learning can be turned into supportive and actionable clinical decisions.

- Catalyze the deployment of ML-based CDS. Although machine learning applications in CDS are impressive, there is lack of systems or tools for machine learning and domain experts to build those applications. These questions will repeatedly be asked: How to access, clean, and use the health data for machine learning development? How to deliver ML-based CDS back to the doctors in real-time? How to operate, monitor, and debug the ML-based CDS once it is deployed? Systems should extricate machine learning and domain experts from

those questions and let them focus on domain-specific problems, e.g., model tuning and improvement.

2.2 Architecture

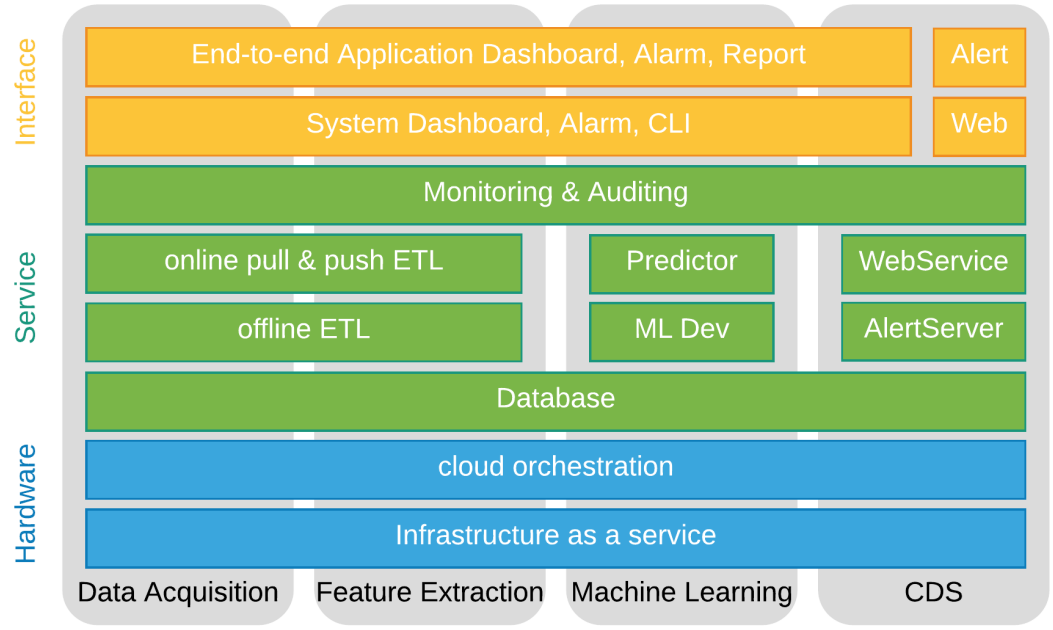


Figure 2.1: The CDS-Stack provides an end-to-end stack across hardware, service, and interface layers for the development and deployment of machine learning-based clinical decision support.

The CDS-Stack provides a full stack, crossing hardware, services, and interfaces, to deploy an end-to-end machine learning-based clinical decision support system on the cloud. A huge advantage of building CDS externally on the cloud is that the CDS-Stack can utilize the latest technology, maximize the scalability and extensibility, and also contribute to the open source community. In contrast, an internal CDS design can

increase the coupling with a specific EHR vendor and make the system more difficult to develop, maintain, and enhance. If we also consider the technical backwardness and isolation of commercial EHR vendors, the design choice of the internal CDS should be avoided. The CDS-Stack aims to support the end-to-end system spanning from four main functionalities, which are data acquisition, feature extraction, machine learning, and CDS.

2.2.1 Hardware layer

The CDS-Stack hardware foundation is based on an infrastructure as a service (IaaS) model to achieve scalable, flexible, and cost-efficient hardware usage for various CDS usage cases. With IaaS, developers can direct access to their servers and storage, just as they would with traditional servers but gain access to a much higher order of scalability. Users of IaaS can outsource and build a "virtual data center" in the cloud and have access to many of the same technologies and resource capabilities of a traditional data center without having to invest in capacity planning or the physical maintenance and management of it. Compared with other two cloud service models, i.e., Software as a Service (SaaS) and Platform as a Service (PaaS), IaaS is the most flexible cloud computing model and allows for automated deployment of servers, processing power, storage, and networking. IaaS gives CDS-Stack developers true control over their infrastructure than users of PaaS or SaaS services.

The CDS-Stack provides cloud orchestration tool to manage the interconnections

and interactions among workflows on the cloud infrastructure. The cloud orchestration tool in CDS-Stack is used to provision, deploy or start services, acquire and assign storage capacity, manage networking, and gain access to specific software on cloud services. It enables service, workload and resource orchestration and can integrate permission checks for security and compliance.

The hardware layer allows developers to allocate all resources and scale them manually or automatically based on the on-demand usage.

2.2.2 Service layer

When considering scalable system design, it helps to decouple functionality and think about each part of the system as its own service with a clearly defined interface. Therefore, the service layer of CDS-Stack defines all the necessary building blocks (services) for the end-to-end ML-based CDSS.

For example, the database service provides the APIs for the data storage and access to both offline datasets (data warehouse) and online data flow (online database). The offline and online ETL components take in charge of data acquisition and feature extraction for historical EHR datasets and online EHR data streams respectively. The raw EHR data will be extracted, cleaned and transformed into a common data model (CDM) and saved into the database. The ML development and prediction components are the core of ML. CDS-Stack does not specify which kind of ML framework to use; the developers can integrate their favorite ML tool as a container in CDS-

Stack, e.g., using R, Python scikit-learn, Spark, or TensorFlow. And the `AlertServer` and `WebService` are the two major components to generate and deliver CDS. The `AlertServer` coordinates the CDS generation when new ETL is done. It triggers the predictor to predict on updated patients and then generates the CDS content and alerts. The main contents of CDS are served as web pages in the `WebService`. Those web-based contents can be seamlessly embedded into the EHR system to server the treatment team. The `WebService` uses a typical memory cache to minimize the latency of requests and can scale up and down based on the service load. Once the new contents and alerts are ready, the `AlertServer` notifies the `WebService` to invalidate and refresh the cache and then push the new alerts to the targeted EHR system. Besides, monitoring and auditing services take the system and user behavior under supervision. Developers can define either system or application metrics and push to a central monitor to visualize, set alarms, and automatically react to the changes in the system.

Creating redundancy in a system can remove single points of failure and provide a backup or spare functionality if needed in a crisis. To handle failure gracefully, CDS-Stack enables the redundancy of its services by default. For example, by default, at least two instances of the `WebService` is running to serve the CDS. If there is only one instance of the web service, then losing that instance means the CDSS is out of service.

2.2.3 Interface layer

On the interfaces layer, the web and alerts are the two types of interfaces to the doctors. The web interface is cross-platform and accessible on any desktop, laptop, or smartphone. Its interoperability allows it is easily embedded into the EHR system. The alert interface is an event-based interface to trigger the treatment team with alerting messages. Those messages can be pushed to the EHR vendor's notification system through its API.

Besides, interfaces for maintenance and monitoring purposes are provided. For instance, the application level dashboard, alarm, and report are provided for the system administrators to monitor the running and performance of the end-to-end application. Developers can establish periodical jobs to generate outcome reports to either their emails or communication channels; they can also add new metrics and visualize them in the dashboard to monitor some key numbers, e.g., current positive cases, number of infected inpatients, etc. System-related dashboard and alarm are used to monitor system performance and debugging. Traditional cloud system monitoring is provided, e.g., monitoring of CPU, memory, IO usages of each instance and group. Some critical service metrics are implemented too, e.g., the duration of ELT, prediction, etc. The developers can customize those metrics and dashboards.

2.3 Implementation

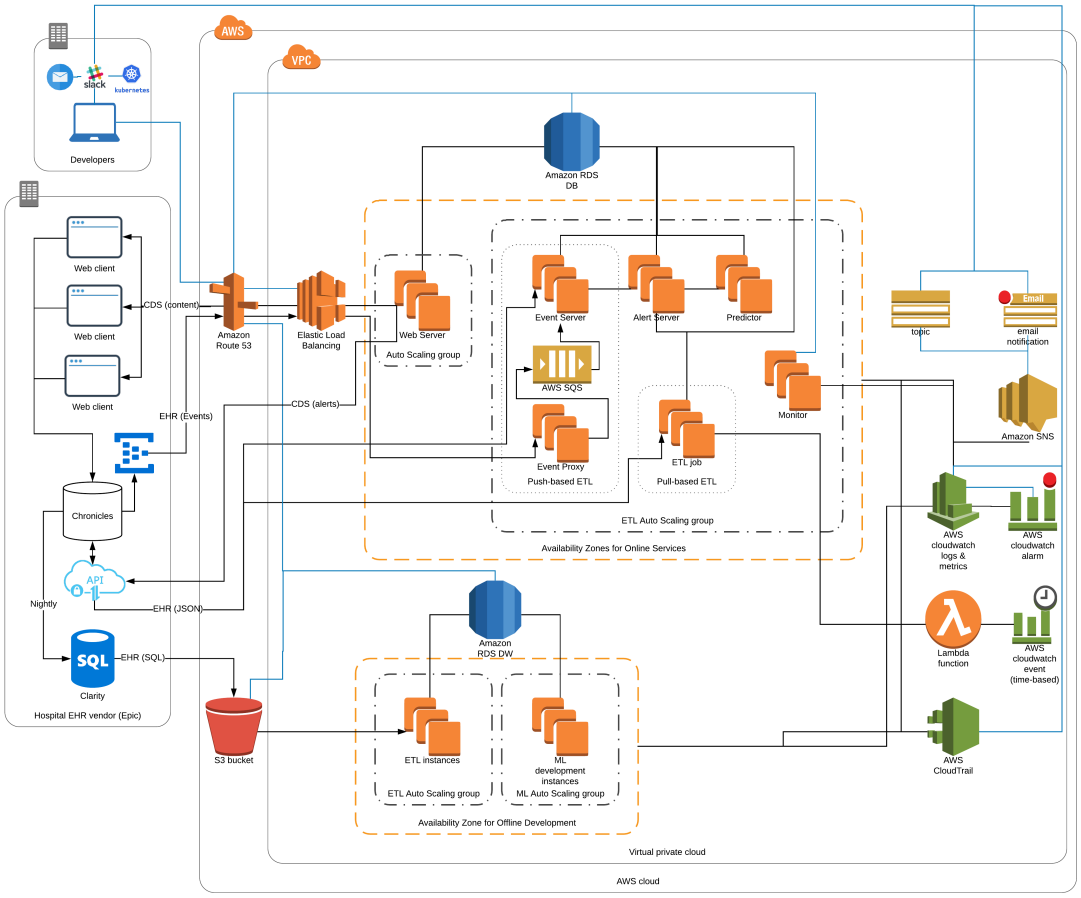


Figure 2.2: The implementation of CDS-Stack on AWS. The black lines stand for the data flows and the blue lines represent the control flows for the developers.

Fig. 2.2 draws the implementation of the CDS-Stack on Amazon Web Services (AWS) and presents the data and control flows in detail. We elaborate five critical components implemented in CDS-Stack as follow, which are cloud infrastructure, end-to-end data pipelines, CDS integration, monitoring, and security.

2.3.1 Cloud infrastructure

As mentioned in Section 2.2.1, the CDS-Stack is built on top of an infrastructure as a service (IaaS) model. AWS, as one of the leading IaaS providers, supplies a range of services to accompany those infrastructure components, including detailed billing, monitoring, log access, security, load balancing, and clustering, as well as storage resiliency, such as backup, replication, and recovery. The CDS-Stack is located in an AWS Virtual Private Cloud (VPC) which is a virtual network dedicated to CDS-Stack AWS account and logically isolated from other virtual networks in the AWS Cloud. The IaaS model provides customizable resource usage (e.g., auto scaling groups) and can "pay as you go" which is much more flexible for the developers comparing with other cloud computing models, e.g., platform as a service (PaaS) or software as a service (SaaS). For instance, The `WebServer` is deployed in an auto-scaling group which can scale up and down based on the load of the web service. Terraform[20] is used in CDS-Stack to codify the configuration of the infrastructure into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

Also, cloud orchestration is achieved via Kubernetes[21] to manage the resources and services on the cloud infrastructure. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized services. In CDS-Stack, for example, all services are implemented as a Kubernetes' *Pod* — the smallest and simplest unit in the Kubernetes object model that you create or deploy. A

Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run. It represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources. For example, the `WebServer` pod is the object to deploy the web service of CDS-Stack. It includes two containers, a Nginx container, and a Gunicorn container. The Nginx one is the HTTP server to host the Gunicorn web application, i.e., a Python web service implemented the CDS application logic. Each pod has a configuration file to define which containers to include (i.e., the docker images), resource usages (i.e., CPU and memory), the number of replicas, and the public URL, etc. This abstraction simplifies and automates the deployment of each service on the CDS-Stack.

2.3.2 End-to-end data pipelines

The end-to-end data pipelines cover both the offline and online data paths from requesting health data from external data sources, extracting features, to running machine learning and finally generating clinical decision support. The "pearls" in this necklace are wrapped as containerized services as well. For example, the data acquisition and feature extraction process are implemented as an ETL service or job. In the offline ETL, the original EHR data is extracted from the historical EHR

database, e.g., Epic Clarity [1], and occasionally or periodically dumped to the AWS S3 bucket. Then, the ETL instance can be established to transform and load them from the bucket to the data warehouse using the AWS RDS. The machine learning development component can be implemented as a containerized service using any ML libraries, e.g., TensorFlow.[19] The training results then can be saved in the data warehouse.

For online ETL, both the pull and push-based protocols are implemented. The pull-based protocol periodically creates an ETL job in the ETL auto scaling group by using the time-based AWS CloudWatch event (similar to a cron job on the cloud). The ETL job will extract all EHRs for current bedded patients in a hospital, including vital signs, medications, lab results and procedures, diagnosis and medical history, clinical notes, etc., and then transform and load the data into the online database using the same pipeline as the offline ETL. In contrast, the push-based protocol will keep maintaining live connections between the EHR event dispatcher and the `EventProxy` service to make sure all live EHR events will be forward to the event buffer in time. The event buffer is implemented as a queue in the AWS Simple Queue Service (SQS) which guarantees all incoming messages will be reliably enqueued and then sent to the consumer (`EventServer`) at least once. The `EventServer` then triggers the ETL process as same as the pull-based protocol. The significant advantage of the push-based protocol is that the ETL can run in a timely fashion for only the patients

[1]The Clarity database is a SQL database for analytics which extracts EHR from the Epic's production database Chronicles every night

who have new measurements. When the ETL is done, whether pull-based or push-based, the `AlertServer` will receive the ETL done message from the ETL instance and then trigger the predictor to run and generate CDS contents and alerts based on the prediction results. For ML developers, they can focus on building the prediction service here and decouple the rest of the application logic into other services.

In the offline pipeline, we use different `dataset_id` and `model_id` to represent different EHR datasets and machine learning models respectively so that developers can compare and evaluate their approaches on different datasets with various machine learning algorithms.

In the online pipeline, we enable three environments which are *test*, *stage*, and *production*. Each of them has their own services, e.g., database and web service. The test environment is connected to the test environment of the EHR system, and developers can do unit and integration tests on the test environment, e.g., new feature extraction and new CDS logic. Both stage and production environments are connected with the EHR's production system but only the CDS from the production environment can be used in the EHR system. Developers can use this stage environment to run end-to-end performance tests, e.g., test a new model for one week before launch. An update on the end-to-end system has to pass the tests on the test and stage environment first, before launching to the production environment.

Chapter 3 elaborates the data schema and ETL design to enable the end-to-end data pipelines, and Chapter 4 describes the two protocols — pull and push — in

delivering real-time CDS in detail.

2.3.3 CDS integration

The CDS-Stack attempts to integrate CDS seamlessly into the EHR vendor's system. It embeds the CDS into the current clinical workflow and interface, so as to minimize the changes of the current workflows and the following educational cost.

The two primary CDS interfaces are CDS alerts and web pages. An alert can be pushed to the right places instantly once a new CDS is produced. For example, an alert can push to the patient list where the nurses and doctors keep checking or display as a notification message to the doctor's phone. The alert service aims to trigger the treatment team and let them investigate the alert in the web pages. The web pages provide a user interface with one-stop service, including the detailed messages to deliver and actions the treatment team can take so that if the physician acknowledges the alert, they can place additional orders or medications without switch context.

In sum, the CDS-Stack applies a hybrid integration approach to deliver CDS back to the EHR vendor's system: the decision support logic is implemented entirely external, i.e., in the CDS-Stack cloud, but the decision support interface is embedded into the EHR vendor's system. This design choice maximizes the flexibility of CDS-Stack in implementing the CDS logic and user interface, and meanwhile, minimize the changes needed for the treatment team to use the CDS in their daily workflow.

2.3.4 Monitoring

It is dangerous to rely on an unmonitored machine learning model for any decision-making. As ML models are deployed, they must be monitored and tuned periodically. Otherwise, ML models may "drift", in which phenomena evolve, but models do not can be catastrophic. For example, Google Flu Trends, which used common search terms as a signal for influenza prevalence, was prominently featured in a 2008 Nature paper, only to later miss the peak of the 2013 flu season by a considerable margin.[22] The monitoring services of CDS-Stack keep collecting metrics, draw dashboards, and trigger alarms for developers to manage production services and debug. The CDS-Stack utilizes AWS CloudWatch and Prometheus[23] to collect metrics in both hardware, service, and interface levels.

Example of dashboards are drawn in Grafana,[24] an open platform for data analytics and monitoring, to display all running status in a central place (see Fig. 2.3). These dashboards can help developers to keep monitoring the hardware usage for each service in the cloud; the application can also setup dashboard to track critical processes, e.g., the data streaming in each ETL. Furthermore, data visualization is included in CDS-Stack too by using Redash[25] to help understand and investigate all the data in the database (see Fig. 2.4). For example, Fig. 2.4a shows the current bedded patients grouped by the hospital; Fig. 2.4b summarizes the number of timestamped features per hour during the latest 48 hours. Fig. 2.4 is an exciting flowchart to show how patients move in the hospital. These figures directly query and visualize

(a) Hardware dashboard.

(b) ETL dashboard.

(c) CDS dashboard.

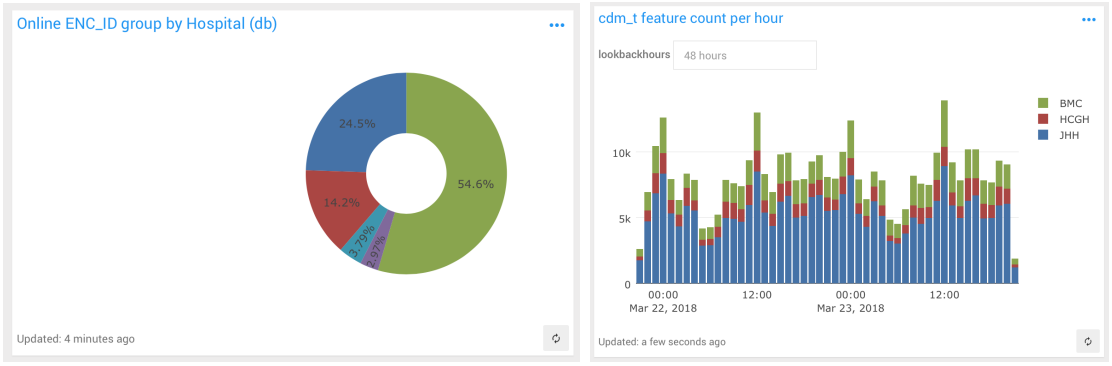
Figure 2.3: The hardware and service dashboards in CDS-Stack

the data from the databases and can be a powerful tool for data exploration.

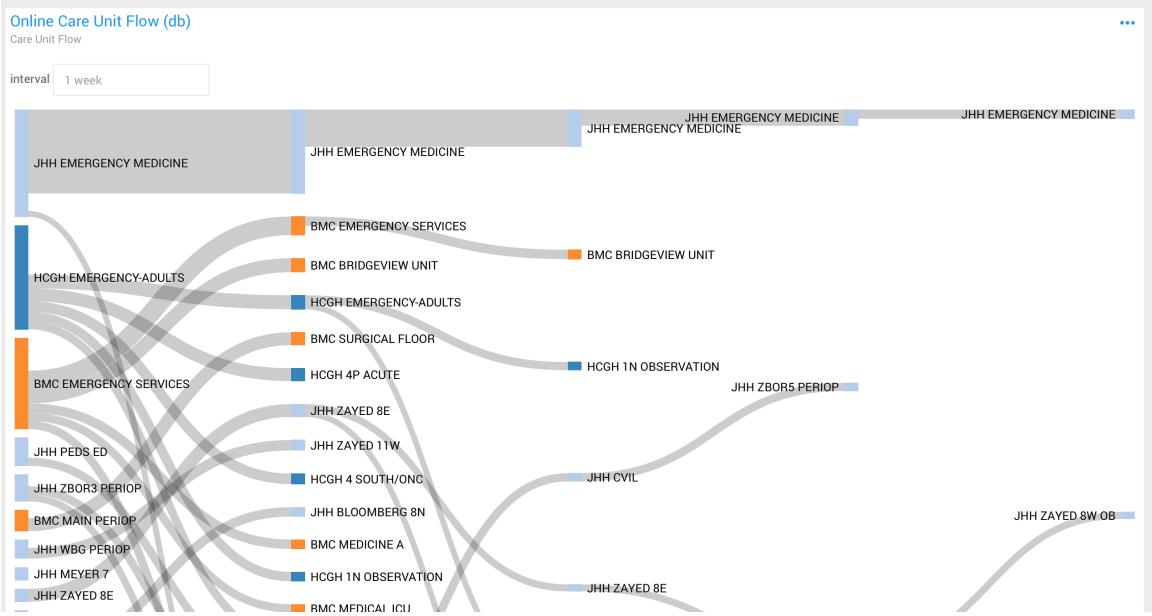
The CDS-Stack alarms are defined using AWS CloudWatch Alarm and configured to deliver to multiple channels, e.g., developers' emails and the developers' channel on Slack, using Amazon Simple Notification Service (SNS). Fig. 2.5 shows both the revoked and active alarms have been delivered to the developers' channel on Slack.

2.3.5 Security

The CDS-Stack architects for HIPAA security and compliance on AWS.[26] In high level, AWS Key Management Service (AWS KMS) is used to manage keys and encrypt PHI. Specifically, there are two aspects that CDS-Stack provides to enable HIPAA



(a) Encounter visualization. **(b)** CDM data visualization.



(c) Care unit flow visualization.

Figure 2.4: The CDS-Stack data dashboards.

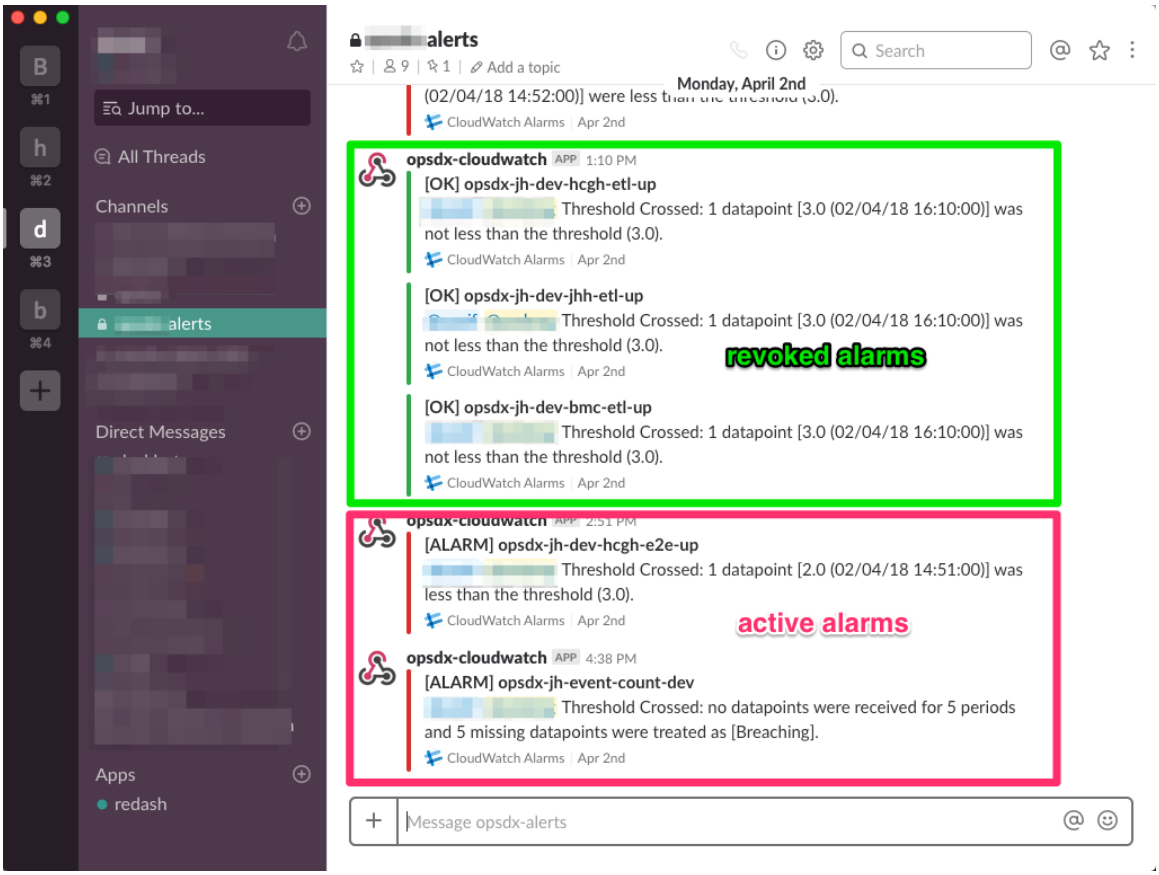


Figure 2.5: The alarms have been fired to the developers' channel on Slack.

security and compliance. One is to guarantee the Personal Health Information (PHI) are always encrypted at rest or in transit. The other is to enable auditing, back-ups, and disaster recovery.

2.3.5.1 Encryption and protection of PHI

The HIPAA Security Rule includes addressable implementation specifications for the encryption of PHI in transmission ("in transit") and in storage ("at rest"). Although this is an addressable implementation specification in HIPAA, AWS requires customers to encrypt PHI stored in or transmitted using HIPAA-eligible services under guidance from the Secretary of Health and Human Services (HHS).

In CDS-Stack, all network traffic containing PHI has been encrypted in transit. For traffic between external sources (such as the EHR vendor) and CDS-Stack, industry-standard transport encryption mechanisms such as TLS are used. When the data is at rest, i.e., in the databases, the data is always encrypted, and the secret is securely stored.

2.3.5.2 Auditing, back-ups, and disaster recovery

HIPAA's Security Rule also requires in-depth auditing capabilities, data back-up procedures, and disaster recovery mechanisms. The services in CDS-Stack contain many features that help customers address these requirements.

The CDS-Stack puts auditing capabilities in place to allow security analysts to

examine detailed activity logs or reports to see who had access, IP address entry, what data was accessed, etc. This data can be tracked, logged, and stored in a central location (e.g., CloudWatch and CloudTrail) for extended periods of time (e.g., one month), in case of an audit. Also, CDS-Stack can further back up the log files into Amazon S3 for long-term reliable storage. Additionally, all CDS contents and alerts generated in CDS-Stack has been sent to the EHR vendor and stored in the EHR database for auditing.

Under HIPAA, covered entities must have a contingency plan to protect data in case of an emergency and must create and maintain retrievable exact copies of electronic PHI. CDS-Stack implements both data and service backup plan on AWS: CDS-Stack keeps backing-up RDS databases every day by default. Amazon S3 also provides a highly available solution for data storage and automated back-ups. By simply loading a file or image into Amazon S3, multiple redundant copies are automatically created and stored in separate data centers. These files can be accessed at any time, from anywhere (based on permissions), and are stored until intentionally deleted. By default, the CDS-Stack launches production instances (Amazon EC2 instances) in multiple Availability Zones to create geographically diverse, fault-tolerant systems, and most other probable sources of downtime.

2.4 Preliminary Results

Starting from 2017, we implemented and deployed the TREWS[7] alert — a sepsis early warning score based alert — using the CDS-Stack to replace current CMS sepsis alert in JHMI. It is the first time that a machine learning-based diagnostic decision support has been deployed into healthcare practice in a hospital, to the best of our knowledge. This section shares the preliminary results we learned from this deployment.

2.4.1 What is sepsis?

Sepsis, a syndrome of physiologic, pathologic, and biochemical abnormalities induced by infection, is the leading cause of death in US hospitals and also a major public health concern, accounting for more than $20 billion (5.2%) of total US hospital costs in 2011.[27] Awareness of sepsis can saves lives. This awareness has led to efforts to ensure that all sepsis patients are treated quickly, including the Centers for Medicare and Medicaid Service (CMS) guidelines for Severe Sepsis and Septic Shock Quality Measures. However, the CMS continues to rely on the Sepsis-2 definition which results in inadequate sensitivity and specificity of SIRS criteria for identifying sepsis. Instead, the retrospective analysis indicated that the new Sepsis-3 definition could be a useful clinical tool. However, because most of the data were extracted from extracted US databases, the task force strongly encourages prospective valida-

tion in the multiple US and non-US healthcare settings to confirm its robustness and potential for incorporation into future iterations of the definitions.[28]

Based on such reality, researchers started to use machine learning methodology to detect sepsis and septic shock. TREWS[7] is one of the best sepsis early warning system among all publications. It identified patients before the onset of septic shock with an area under the ROC (receiver operating characteristic) curve (AUC) of 0.83 [95% confidence interval (CI), 0.81 to 0.85]. At a specificity of 0.67, TREWScore achieved a sensitivity of 0.85 and identified patients a median of 28.2 [interquartile range (IQR), 10.6 to 94.2] hours before onset. It can be a powerful alert to replace current rule-based CMS sepsis guideline.

2.4.2 TREWS deployment

TREWS was initially developed on the MIMIC (Multiparameter Intelligent Monitoring in Intensive Care)–II Clinical Database.[29] To build and deploy it into hospitals in JHMI, TREWS has evolved with new ideas[30] and re-trained from the JHMI datasets, specifically, a three-year EHR dataset of Howard County General Hospital (HCGH) stored in the CDS-Stack data warehouse. The CDS-Stack enables the offline analysis and the online pipeline to build and deliver TREWS alerts to hospitals in JHMI. The latest version of TREWS showed it could identify 80% of the sepsis cases of 0.40 positive predictive values (PPV). In contrast, the current CMS definition only identified 49% of patients any time prior or within half hour after the onset of sepsis.

And the PPV of CMS alerts (assuming alerting every time criteria is met) was less than 0.09.

Using the CDS-Stack, the TREWS alert has been deployed in HCGH since November 2017. From November 2017 to the end of December 2017, 13,416 patient encounters have been monitored in the background, and 845 patients among all have shown positive on TREWS alerts. The preliminary results are shown as follow:

- 51.5% of TREWS alerts had a user response.
- 60% of alerts got one page view at least.
- Of alerts with a user action, 59.0% confirmed infection, 41.0% entered no infection suspected, and 1% used manual override without entering SOI.

The user engagement has been shown to be much better than the previous CMS alerts which were usually ignored by the treatment team. In this deployment, the CDS-Stack supported TREWS to 1) maximize feature engineering code in both offline training and online prediction; 2) keep feeding the TREWS model with online EHR data; and more importantly, 3) continue monitoring its performance, including accuracy and user engagement.

2.5 Discussion

The CDS-Stack provides an open-source toolkit to relief the pain of developing and deploying machine learning-based CDS into clinical practices from scratch. Com-

pared with other CDSSs[31–33] which generally adopted the SaaS model, CDS-Stack maximizes the flexibility and reusability for ML practitioners to implement new ML-based CDS and extend it into new medical institutes. The machine learning and domain experts can focus on feature selection and tuning of the machine learning algorithms instead of "plumbing", i.e., integrating various components together into production. To our best knowledge, CDS-Stack is the first open-source cloud-based solution that speeds innovation on ML-based CDS.

So far, the CDS-Stack is still the first step to catalyze clinical innovations using machine learning. It has several limitations. First, the CDS-Stack was implemented and evaluated only with the leading EHR vendor in US, Epic system, within JHMI currently. It will be beneficial to extend and try it with other EHR systems and medical institutes. Second, current clinical outcomes were implemented manually in CDS-Stack, a library of standard clinical outcome metrics is not available now. Such a library, including standard clinical metrics like length of stay, mortality, readmission rate, etc., can be used to evaluate ML-based CDS in the long run and provide a meaningful clinical datasheet for CDS benchmarking. Third, the web-based CDS interface only supported one theme or style right now. A system to switch and compare multiple interface design can be valuable to improve the human-computer interface.

Chapter 3

CDM: Transforming EHRs to Common Data Model

3.1 Motivation

"It is a capital mistake to theorize before one has data."

– Sherlock Holmes, "A Study in Scarlett" (Arthur Conan Doyle).

The premise of using ML in CDS is to be able to convert EHRs into the right data format for ML usage, e.g., training or prediction. However, EHRs cannot be directly consumed by machine learning algorithms.

First of all, the generic format of ML dataset, called *data frame*, is a tabular dataset with each row as a sample and each column as a specific feature or measurement, while, current EHR data is heterogenous and dispersed on multiple different

places with various formats. For example, The Epic Clarity database, one of the most adopted historical EHR database, has thousands of tables and the EHR data like vital signs, lab results, clinical notes, etc. is stored at different tables with different attributes. Furthermore, if you want to access online EHRs in real-time, you have to request the endpoints from the Epic's interconnect web service using SOAP or REST APIs, or FHIR web services which is also based on REST APIs. Multiple web requests are needed and the received data format changes to either JSON or XML.

Second, the success of ML development and deployment rely on high-quality data and data consistency. However, since current EHRs are not quality guaranteed and the interoperability is quite low among different sites and EHR vendors, A data cleaning and preprocessing is essential before severing them to ML algorithms.

Third, even though some EHR vendors start to build EDWs (enterprise data warehouses) and try to share EHRs in an easy-to-use format, their data schema is not specific for ML usage. Machine learning scientists or engineers still need to operate multiple complex queries and calculations to generate a dataset ready for use, including data cleaning, table join, fill in missing values, feature extraction, and so on.

To our best knowledge, there is no such a tool specifically designed to convert EHRs for ML usage. Therefore, we introduce CDM (common data model) — a simple and generic database schema for machine learning usage. The CDM contains an entity–attribute–value model (EAV) to represent health data in a simple form and

a data frame format for machine learning algorithms to use directly. To well abstract the conversion from EHR to CDM, An ETL engine is designed and implemented to execute this process. The ETL engine is a modularity framework in order to improve the maintainability and scalability and maximize the code reusability. CDM unifies the data format on both the offline and online databases. The benefit is that once the online CDM data pipeline is developed, the ML models developed from the offline CDM dataset can be directly deployed into the online environment.

3.2 CDM database schema

CDM is designed as a database schema in relational (SQL) database. We choose SQL instead of NoSQL because we need a robust store with enforced data integrity and transaction support. Only an SQL database will (currently) satisfy those requirements. The core CDM tables is shown in Fig. 3.1 and described in Table. 3.1. Note that both the data warehouse and online database of CDS-Stack use the same CDM schema and the only difference is that the online database has no `dw_version` table and `dataset_id` column in these tables because the online database only contains a single data source.

The main design principle of CDM is to keep it simple and easy to adopt:

Only simple query is needed. Compared with the thousands of tables in the Clarity database, CDM only contains less than ten main tables. All features can be

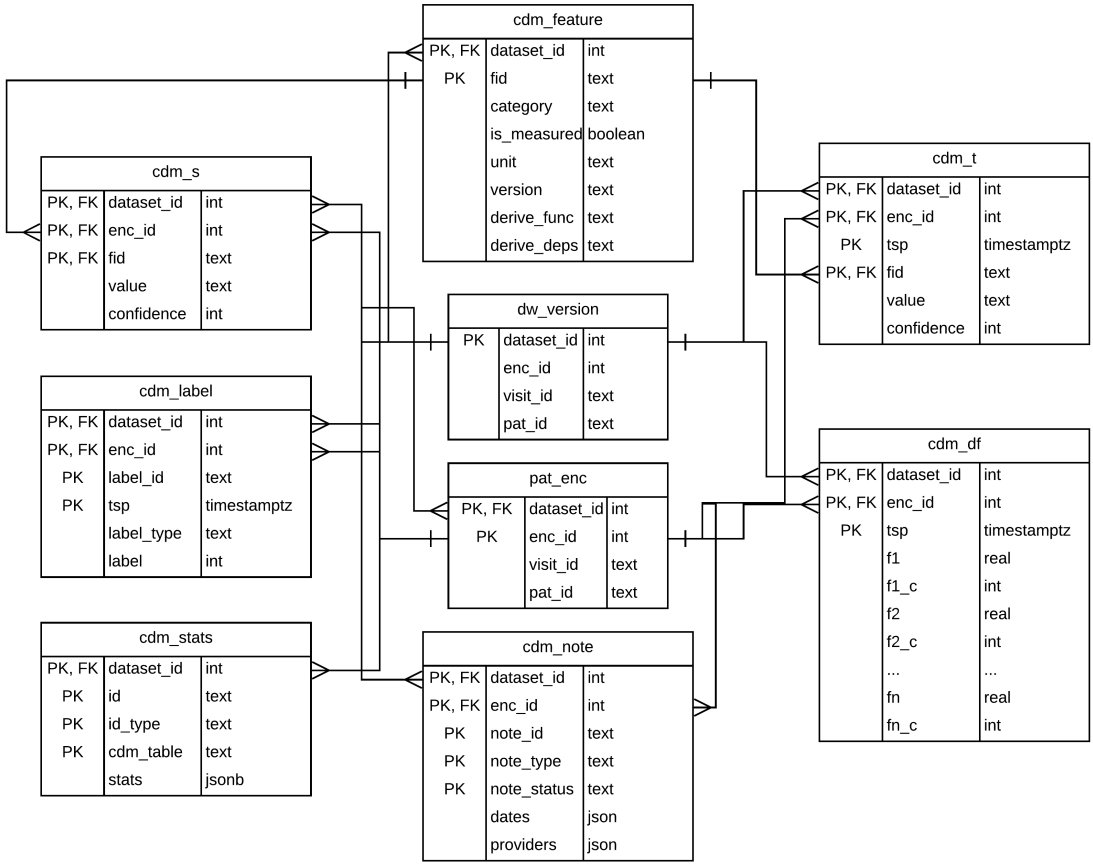


Figure 3.1: The core CDM database schema. The `cdm_s`, `cdm_t`, and `cdm_note` tables represent EHR using the EAV model and the `cdm_df` table is the data frame to store a complete dataset for ML usage.

Table	Description
dw_version	The table list all datasets in the data warehouse.
pat_enc	The main patient table. The enc_id (encounter ID) is the internal id of an inpatient visit. The pat_id and visit_id stores the original patient ID and visit ID respectively.
cdm_feature	The CDM feature dictionary. It list the feature's category (i.e., which CDM table it locates), data type (e.g., Integer, Real, Boolean, JSON, String), and either raw measured feature or derivative features. If it is a derivative feature, it list its derive function and depending features.
cdm_s	The CDM table stores static feature values, i.e., those keep stable in an inpatient visit, e.g., age, gender, medical history, etc.
cdm_t	The table stores timestamped feature values, e.g., vital signs, lab results, medication administration, etc.
cdm_df	The data frame table. The primary keys are dataset_id, enc_id, and tsp. The rest columns are the measurements to be used in machine learning. The columns ending with _c is the confidence value of that measurement, which indicates how does the value generated, e.g., originally load from EHR source, transformed, filled-in.
cdm_note	The table stores clinical notes.
cdm_label	The table stores generated labels with version control.
cdm_stat	The table stores statistical reports of CDM features.

Table 3.1: The description of the core CDM Tables.

easily located by query the `cdm_feature` table.

Can be directly consumed by ML algorithms. A ML algorithm can directly load `cdm_df` table to start training or prediction. In addition, a simple join query can be used to combine some static feature values, e.g., age and gender from `cdm_s` with `cdm_df`.

Supports various EHR data types. The data type of the `value` field in `cdm_s` and `cdm_t` is `text` which supports flexible data types. Even some complex data type, e.g., some events with multiple attributes, can be represent as `JSON` string to store in this field.

Allows scalable read and write. All the CDM data tables can be partitioned by `dataset_id` and `enc_id` in either read or write operations. It is highly parallelizable for reading and writing to those tables.

3.3 ETL: transforming EHR to CDM

How to transform original EHRs into CDM? We describe this process as an ETL process. ETL is short for extract, transform, and load, which are the three steps to pull data out of one database and place it into another database. Specifically, the ETL in CDS-Stack stands for the process of extracting EHR data sources, e.g., databases or REST APIs, transforming, and loading into a CDM database.

The concept of the ETL process is drawn in Fig. 3.2. The ETL first extracts the

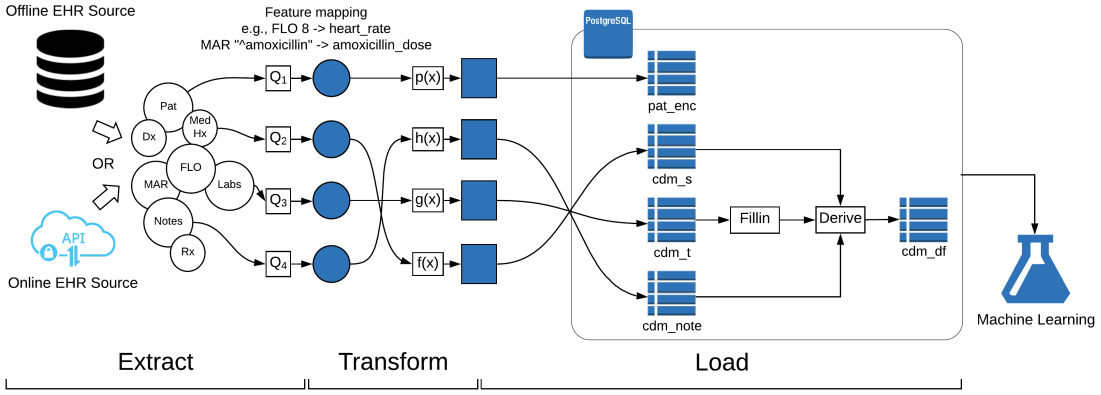


Figure 3.2: The ETL data integration from EHR to CDM.

targeted EHRs from either historical EHR databases or online APIs with a feature mapping configuration. The feature mapping configuration is predefined with specific queries (e.g., Q_1 in Fig. 3.2) to map a particular external EHR to a feature ID (`fid` defined in `cdm_feature`). A transform function is specified for each `fid` to transform the values (including unit conversion, deleting the dirty or coarse data and data filtering) and then convert and load to the EAV models, i.e., a record in `cdm_t`, `cdm_s`, or `cdm_note`.

The last but not least step is to create the data frame in `cdm_df`. This step includes two sub-steps: `fill-in` and `derive` as illustrated in Fig. 3.3. The fill-in process loads raw measured feature values from `cdm_t` into `cdm_df` and meanwhile fills in missing values using a fill-in function, e.g., carry on the last value. For example, Fig. 3.3 imports two raw features `spo2` (i.e., SpO2) and `hr` (i.e., heart rate). Their measurements have shared timestamps (`tsp`), but there are timestamps when only one measurement exists, e.g., the first row and the last row in this case. The fill-in

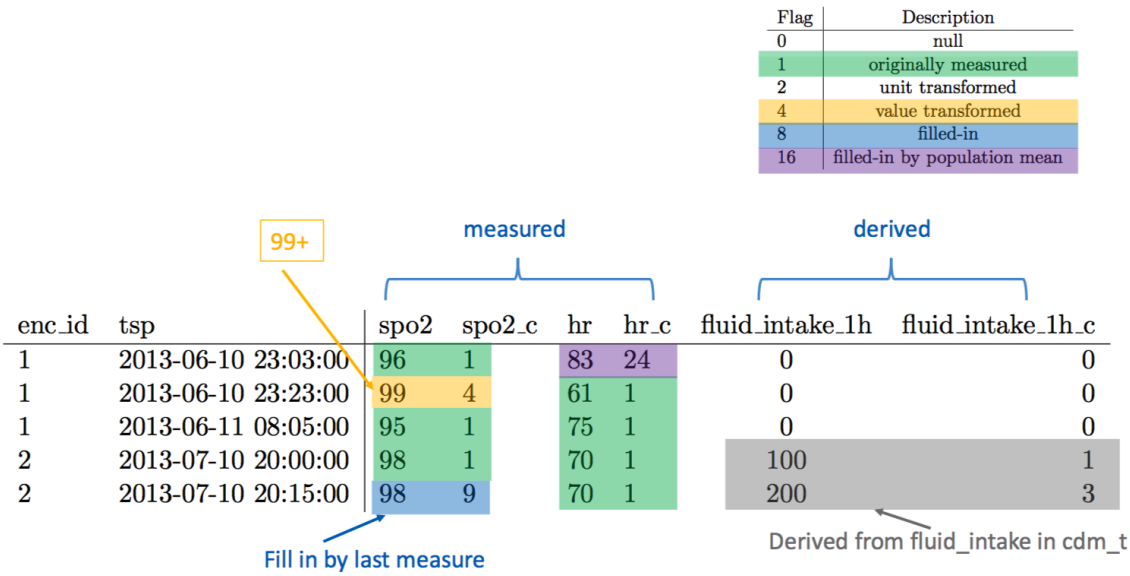


Figure 3.3: An example of generating `cdm_df` table using fill-in and derive process

function is to make sure those gaps can be filled in based on some rules. For example, carry on the last `spo2` value 98 into the last value; for those without previous value, a population mean can be used, e.g., the first heart rate value 83. The confidence values, e.g., `spo2_c` and `hr_c` is used to track the source of those values, either originally measured, unit transformed, or filled-in by last value or population mean.

Once all the missing values are filled in for raw measurements, it's easier to generate derivative features based on them. Taking SIRS as an example, mentioned in Table 1.1, it can be easily calculated if all raw input measurements are ready for each timestamp. Fig. 3.4 shows all the features needed to generate a derivative feature called `septic_shock`, including raw measurements (i.e., the leaves without any inbound arrow) and other derivative features. This dependency graph is a directed acyclic graph (DAG) which has at least one topological ordering. The topological

sorting can be used to schedule those derive functions based on their dependencies.

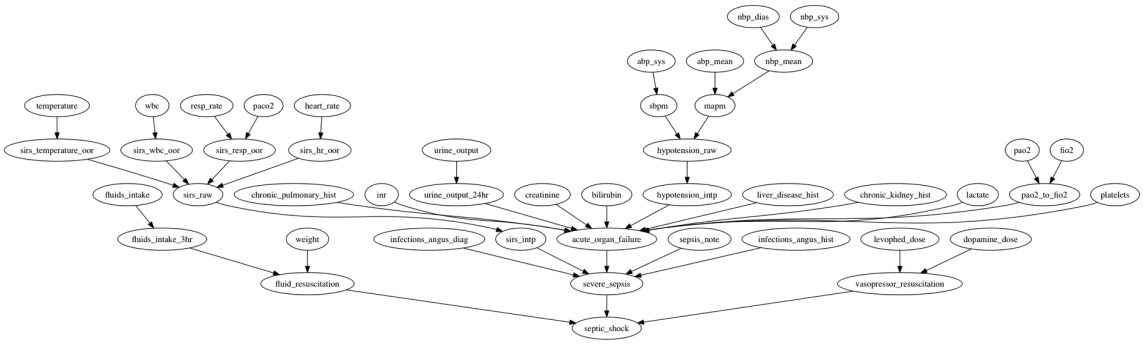


Figure 3.4: An example of dependency graph for septic shock

After the fill-in and derive processes, a complete data frame is generated, which means all raw and derivative features are generated with no missing value.

3.4 The ETL engine: the framework to build ETL pipelines

Section 3.3 describes the concept of the ETL process to turn EHR into CDM and in this section, we introduce the ETL engine to help build such an ETL pipeline and handle both dependency resolution and parallelism.

The ETL engine is implemented using `Asyncio`, the asynchronous programming model introduced after Python 3.4. `Asyncio` is a library to write asynchronous applications. It is the most efficient way to implement a network server having to handle many concurrent users. Here, the ETL engine uses `Asyncio` to handle concurrent IO

tasks, which are heavy on the database queries and HTTP requests. Even though it won't improve latency since await-ed functions still have to wait for IO, while the await-ed functions wait for IO, control is automatically returned to the event loop so other code can run. In contrast, serially making those requests and waiting for responses would be painfully slow. Also, compared with parallel computing using threads, `Asyncio` adopted a radically different solution for race conditions. Code written with `Asyncio` is less error-prone: by just looking at the code, it is possible to identify which parts of the code are under our controls and where the event loop takes over the control flow and can run other tasks when our task is waiting for something.

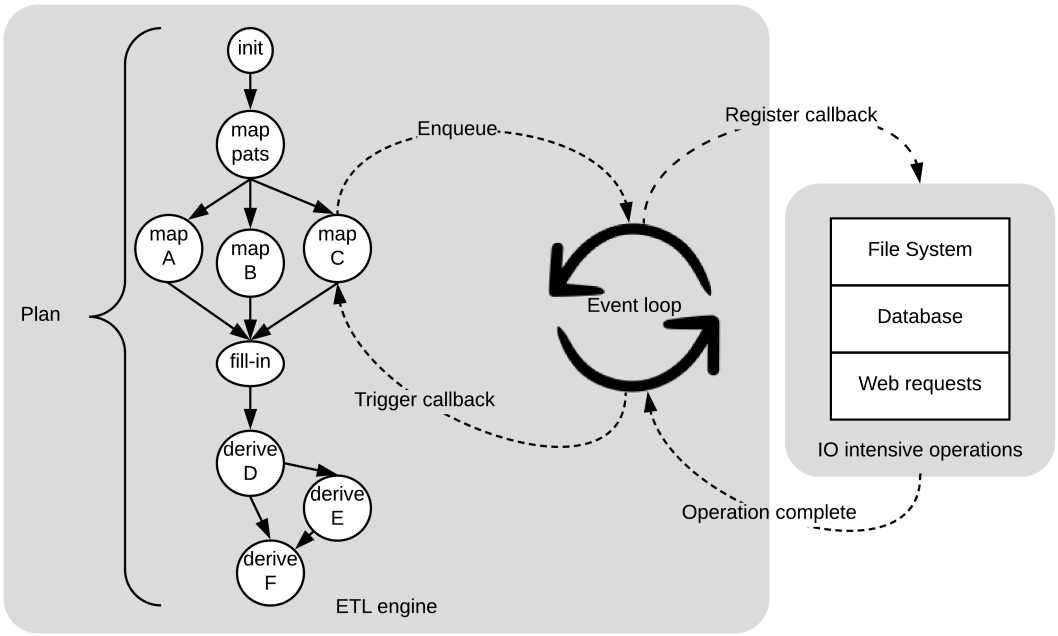


Figure 3.5: The ETL engine. The event loop schedules the asynchronous tasks from the plan based on the dependency graph. While some tasks are waiting for IO intensive operations, the event loop takes over the control flow and is able to run other tasks.

As shown in Fig. 3.5, the ETL engine maintains an event loop and a multi-processor pool to execute a DAG of tasks, called a *plan*. When the engine starts, or every time a task is completed, it enqueues all ready tasks, i.e., the pending tasks without any incomplete depending task. A task is defined as a coroutine with inputs including the engine's context (containing the database pool and engine log), and the outputs from its depending tasks. Then once it is completed, its output will be the

```
async def db_query_task(ctxt, *inputs):
    '''
    @ctxt: the context of this ETL engine
    @input: the outputs from its depending tasks
    '''
    query = generate_sql_query_from_input(inputs)
    async with ctxt.db_pool.acquire() as conn:
        # while waiting for the DB operation
        # the event loop can operate other tasks
        results = await execute_db_query(query)
        return results
```

Figure 3.6: An example ETL task to query the database.

part of the inputs of its dependent tasks. Fig. 3.6 shows a simple code example of a database query task. Note that while this task is waiting for executing the database query, the event loop can operate other concurrent tasks.

Therefore, an ETL pipeline can be represented as a plan and use the ETL engine to execute the plan in parallel. The plan, specifically, defines feature mapping process as a group of tasks and use the semaphore to control the maximum number of concurrent tasks. For fill-in and derive processes, since they are implemented as SQL queries, we partition the queries by `enc_id` so that each task can be executed as multiple

concurrent DB queries orthogonally. To avoid any possible conflicts or deadlocks since fill-in and derive functions operate on the same table, we configure the event loop to allow only one such task a time, i.e., semaphore = 1. Alternatively, another option is to adopt the column-wise parallelism, i.e., executing fill-in function column by column and then executing multiple derive functions simultaneously. However, since The PostgreSQL database (and most of the SQL databases) are row-oriented and use multi-version concurrency control (MVCC), column-wise parallelism can cause two problems: 1) changing one item in a row means to create a whole row with a new version so update one column in a table means the whole table needs to be recreated; 2) bulk inserting or updating different columns but the same rows from multiple processes may cause conflicts and even deadlocks, which may not be scalable or even be the opposite. The parallel performance is evaluated in Section 3.5.

It is also worth mentioning that this ETL engine framework is used on both offline dataset ETL in DW and online data pipeline. The only difference between the online and offline ETL pipelines is that since the online pipeline extracts EHR from web services, the feature mapping functions are specifically implemented to run web requests instead of database queries. But the rest of the tasks, e.g., fill-in and derive functions, are reusable.

3.5 Evaluation

To understand the performance of "read" and "write" on CDM datasets, this section evaluates the scalability of the ETL engine in performing the three major ETL processes: feature mapping, fill-in, and derive. Two parallel approaches, row-wise and column-wise parallelism, are evaluated in the CDM data warehouse using the JHMI CDM datasets. In the end, this section also evaluates the query performance of the CDM datasets.

3.5.1 Experiment setup

In this evaluation, both the DW and ETL machines used the AWS m4.2xlarge instances which have 8 vCPUs, 32 GB memory, and 1,000 Mbps maximum bandwidth. We used the historical EHRs stored in the Clarity database from JHMI to evaluate the ETL performance. Specifically, we dumped the clinical related EHRs from the three main hospitals in JHMI (as shown in Table 3.2) to the DW and measured the performance of converting them to CDM. The features used in this evaluation are list in Table 3.3 which are the key features we used to develop TREWS. The sizes of the CDM datasets are list in Table 3.4.

Hospital	HCGH	JHH	BMC
Time Range	2014/04/20 to 2017/04/20	2016/01/01 to 2017/06/30	2016/01/01 to 2017/01/01
Inpatient visits	223K	574K	141K
Flowsheet Rows	3795 MB	6060 MB	1766 MB
Lab results	551 MB	7672 MB	1107 MB
MAR	491 MB	2894 MB	809 MB
Procedure Orders	699 MB	3996 MB	989 MB
Notes	8550 MB	5576 MB	5059 MB

Table 3.2: Historical EHR extracted from the Clarity database of JHMI. HCGH, JHH, and BMC are the three main hospitals of JHMI.

Category	Count	Examples
ADT	1	Arrival time
Demographics	5	Age, gender, admission time
Diagnoses	30	Dx of septic shock, sepsis, pancreatic cancer, etc.
Flowsheet	18	Weight, heart rate, fluid intake, etc.
LDAs	1	Catheter
Labs	29	WBC, lactate, sodium, etc
Medical History	31	Hx of septic shock, sepsis, pancreatic cancer, etc.
MAR	140	Amoxicillin, Penicillin, Propofol, etc.
Notes	10	sepsis, UTI, pneumonia sepsis notes, and so on
Order Procedures	12	BiPAP, CPAP, EKG procedure, etc
Problem List	34	The problem list including septic shock, sepsis, etc.
Derivative features	45	SIRS, SOFA, severe sepsis, septic shock, etc.

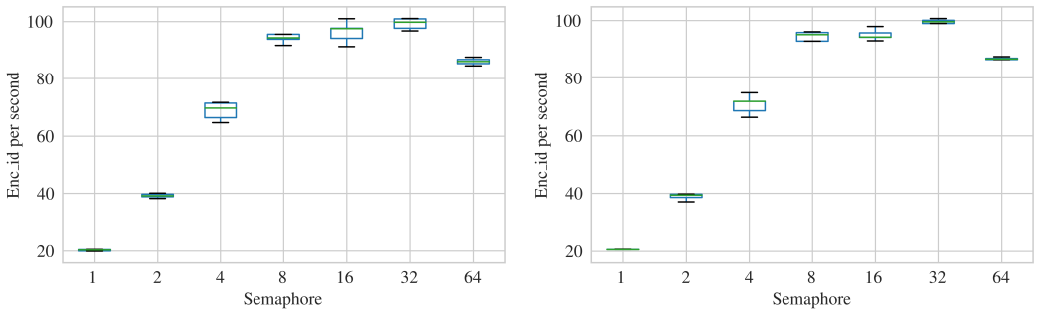
Table 3.3: CDM features used in the evaluation.

Dataset	HCGH	JHH	BMC
Number of enc_ids	223,945	574,506	141,884
Number of rows in cdm_s	1,352,121	2,914,305	726,114
Number of rows in cdm_t	29,095,535	48,783,459	13,920,056
Number of rows in cdm_df	4,439,429	6,462,794	2,098,752
Number of rows in cdm_note	3,346,552	1,113,181	1,063,274

Table 3.4: The sizes of the CDM datasets in DW.

3.5.2 Feature mapping

We first evaluated the process of feature mapping, which includes extracting all the measured features (i.e., except derivative features) from EHR staging tables in DW, and then transforming and loading to the `cdm_s`, `cdm_t`, and `cdm_note` tables. We defined each feature mapping task as an ETL task in the ETL engine and measured the speed-up of the throughputs (i.e., number of enc_id per second) while increasing the number of semaphores and processors of the ETL engine. The number of semaphores controlled the maximum number of concurrent tasks executed in the ETL engine, and the number of processors specified the maximum number of CPUs used in the processor pool.



(a) One process **(b)** Two processes

Figure 3.7: The performance of parallel feature mapping

The performance is illustrated in Fig. 3.7. We set the number of processes from one to two and the semaphore from 1 to 64, i.e., the number of concurrent tasks in the event loop. For example, that semaphore is equal to 1 means all feature mapping tasks have to execute sequentially. The y-axis in the figure shows the throughputs

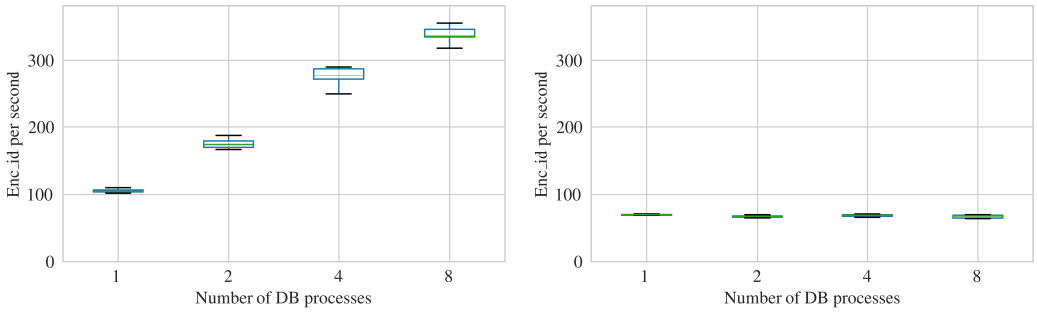
(enc_id per second) of different settings. First, notice that doubling the number of processes used in the ETL engine did not improve the performance for this IO intensive process, i.e., one CPU core was sufficient to handle all computations in the ETL instance. Then, we can see that the throughput scaled from 20 enc_id/s with one semaphore to 100 enc_id/s with 32 semaphores. The throughput converged at eight semaphores because it uses all the cores in the DW instance to run the extraction and loading. The throughput slightly increased from 8 to 32 due to the various amount of loading size among features, i.e., having more concurrent tasks can keep the database always busy. However, once the semaphore increases to 64, the performance dropped due to the increased overhead of context switches.

3.5.3 Fill-in process

Secondly, we evaluated the fill-in process using two parallel approaches: row-wise vs. column-wise. As mentioned in Section 3.4, the fill-in process is fully implemented as SQL queries on the single `cdm_df` table. We hypothesize that row-wise parallelism allows all workers run orthogonally to maximize the speed-up using multi-cores; instead, even though theoretically the column-wise approach can partition the features independently, in practice, the fundamental data structure used in Postgres database tables may mainly reduce the gain from the parallelism.

Fig. 3.8 shows that row-wise fill-in function scaled from 100 enc_id/s to nearly 350 enc_id/s while increasing the number of DB workers from 1 to 8. Instead, the column-

wise fill-in function did not speed up with the increasing number of DB processes. Even when the number of DB processes is one, row-wise parallel is faster than column-wise parallel because the former only needed to traverse and update each row once while the latter needed to traverse and recreate each row N times, here N is the number of columns.



(a) Row-wise **(b)** Column-wise

Figure 3.8: The performance of parallel fill-in function: row-wise vs. column-wise

3.5.4 Derive process

Thirdly, similar to fill-in function, the two parallel approaches – Patient-wise and function-wise – were evaluated here. To execute the DAG of derivative features, the patient-wise approach partitioned the `cdm_df` table into groups of patients and derived features one by one; alternatively, the function-wise derive multiple independent features simultaneously. We hypothesize that the patient-wise approach can parallel orthogonally, but the function-wise can slow down due to the conflicts or even deadlocks while accessing the same rows in the same Postgres table from different

processes.

Fig. 3.9 shows the patient-wise derive process could scale from 13 enc_id/s to 56 enc_id/s, which was over four times improvement. While instead of scaling up, the throughput of the function-wise approach slightly dropped from 13 enc_id/s to 7 enc_id/s, i.e., less than half of the throughput, which was due to the conflicts and deadlocks occurred among concurrent derive functions.

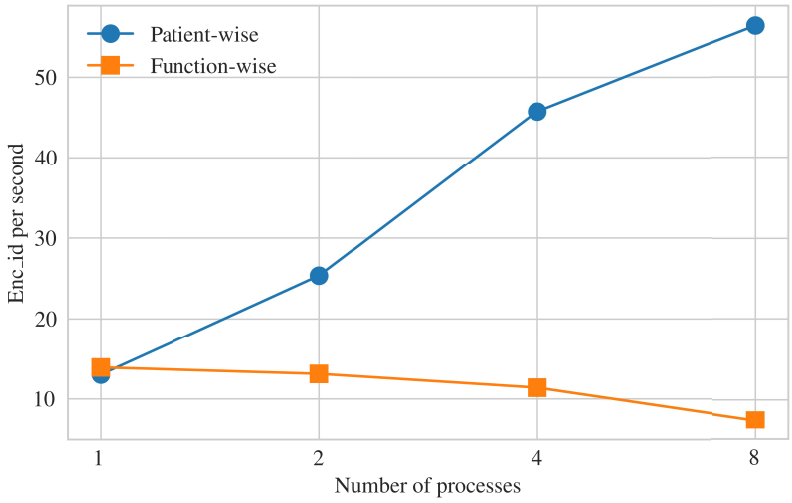


Figure 3.9: The performance of parallel derive process: patient-wise vs. function-wise

Based on the throughput benchmark of the above three steps, creating one of the largest CDM dataset, i.e., JHH, only requires around 5 hours. Compared with the non-parallel approach which needs around 22 hours, it can save 77% of the ETL time.

3.5.5 CDM queries

Besides the performance of writing data into CDM data warehouse, we also evaluated the reading performance, i.e., CDM queries, using a single processor. We attempt to show the duration for reading data from an entire CDM dataset and the throughput (enc_id/s) as well.

Query	Time (s)	TX (enc_id/s)
`SELECT * FROM pat_enc WHERE dataset_id = x`	3.0	191.5K
`SELECT * FROM cdm_s WHERE dataset_id = x`	24.5	23,4K
`SELECT * FROM cdm_t WHERE dataset_id = x`	2,066.3	278.0
`SELECT * FROM cdm_df WHERE dataset_id = x`	1,618.2	355.0
`SELECT * FROM cdm_note WHERE dataset_id = x`	132.2	4345.7
`SELECT * FROM calculate_historical_criteria(pat_id)`	8.1	N/A

Table 3.5: The mean execution time and throughput of example SQL queries on the JHH dataset (using only one DB worker)

Table 3.5 shows the performance of some typical queries using one DB processor or connection. For example, the average execution time to extract all rows from the JHH data frame is 1,618.2 seconds (less than half an hour) or 191.5K enc_id/s. The last query example — `calculate_historical_criteria()` — calculate the CMS sepsis criteria at every hour during the time of hospitalization for one patient. It requires 8 seconds in average to generate one patient's hourly report using one DB worker. We will discuss the solution to further improve those queries in Section 3.6.

3.6 Discussion

CDS-Stack introduced a simple and homogenous data model — CDM — as the data layer of ML core functionalities. It can dramatically reduce the high complexity of clinical data and the ETL engine could accelerate the development of both offline and online data pipelines for new ML-based CDS. In addition, to extend an ML-based CDS to new hospitals or institutes, most of the ETL pipeline can be reused, and the developers only need to adjust the site-dependent part, i.e., feature mappings.

The CDM design and implementation have several limitations. First, the current data storage is implemented and evaluated on a single database which is limited by the total number of CPU cores and the maximum IO bandwidth. This will be a bottleneck for both write and read especially for DW when more and larger datasets have been created.

Second, the current CDM feature definition follows an ad-hoc or application-specific fashion. This can be a drawback when one CDM feature definition is going to support multiple applications. Creating and maintain a standard clinical feature dictionary can be beneficial for extending the CDM features to broader usage. We started the first step by open sourcing the CDM features we created for sepsis early detection.

Overall, one of the future directions to improve CDM is to implement it in a distributed and column-orient database. Since the CDM schema is highly scalable on the `enc_id`, the dataset can be partitioned in multiple database machines and

CDM queries can be run concurrently without space and bandwidth constraints. In addition, a column-orient database should contribute to the derive functions, i.e., each derive function can only update one particular column without affecting other columns, which is impossible to do in a row-orient database. AWS Redshift[34] and Google Bigquery[35] can be two options to implement future CDM data warehouse.

Chapter 4

Delivering Real-time Clinical Decision Support

"I am aware that success is more than a good idea. It is timing too."

– Anita Roddick

4.1 Introduction

Once a machine learning model is developed and evaluated in the CDM data warehouse, the last mile of the end-to-end CDSS is to set up a production system to deliver the CDS to the treatment team. Four major steps can be involved in this stage, shown in Fig. 4.1:

- ETL: The ETL engine runs the pipeline of web requests, transform functions,

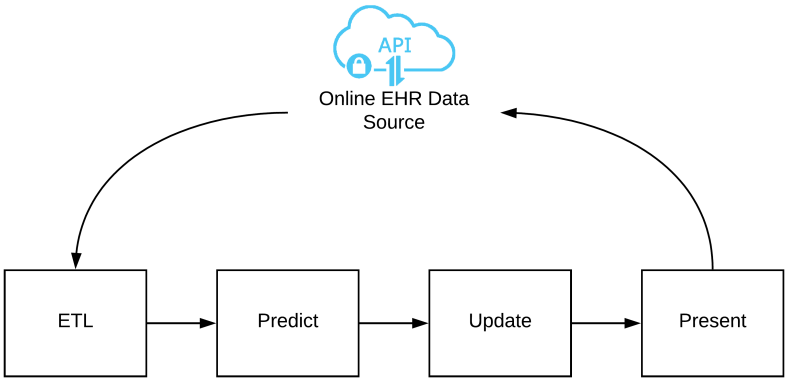


Figure 4.1: Major steps to deliver online diagnostic decision support.

and loading functions to convert EHR to CDM database.

- Predict: the system requests the online machine learning service (predictor) to make predictions based on the latest patient data.

- Update: the system updates all data models needed to show to the users, e.g., prediction results, related clinical criteria and explanation.

- Present: the system presents the updated CDS back to the EHR vendor in form of web pages and notification messages.

How to trigger and run this pipeline in a live mode? If we consider the CDSS as a client-server system between the CDS-Stack and the EHR vendor, then the *pull* and *push* are the two types of protocols to communicate between this client-server system. In pull protocols, the client (i.e., the CDS-Stack) periodically connects to the server (i.e., the EHR vendor), checks for and gets (pulls) recent EHR data and then run the rest of the pipeline. In this mode, the client repeats this whole procedure to

get updated about new EHR data. In push protocols, the client opens a connection to the server and keeps it constantly active. The server will send (push) all new events to the client using that single always-on connection. The difference is that in push protocols, you get new events instantly (such as a new patient admission, a new lab result, etc.). But you may experience a time delay in pull protocols depends on the interval between two online pipelines. We elaborate the implementations of pull and push-based online data pipeline in Section 4.2 and evaluate their performance in Section 4.3.

4.2 Implementation of the Pull- and Push-based Pipelines

The CDS-Stack includes both pull-based and push-based data pipelines to enable real-time online clinical decision support.

The pull-based online data pipeline, as shown in Fig. 4.2, is triggered periodically by a timer with a certain time interval, 15 minutes by default. The timer is implemented as an AWS Cloudwatch event, and it periodically fires an AWS Lambda function which initializes an ETL job at one of the computing instances in the production auto-scaling group. The ETL job runs an ETL engine to extract EHR from the REST API of the EHR vendor [1], transform EHR to CDM, and load it to the

[1]We choose the REST API because it is the most well-maintained and used interface in JHMI's

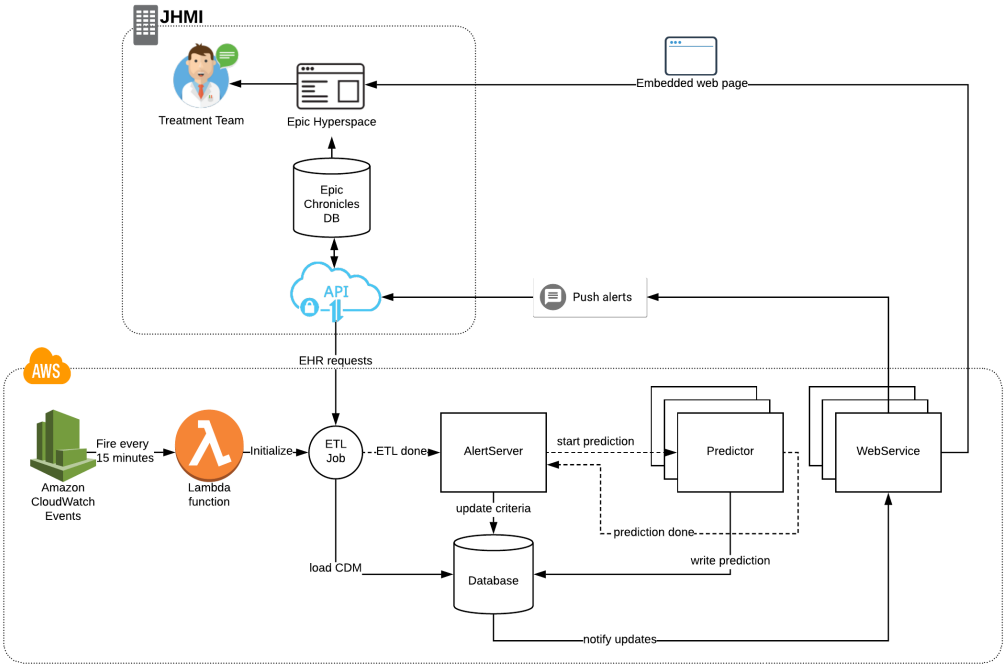


Figure 4.2: The implementation of the pull-based online pipeline, which is triggered by a periodical timer event from AWS CloudWatch.

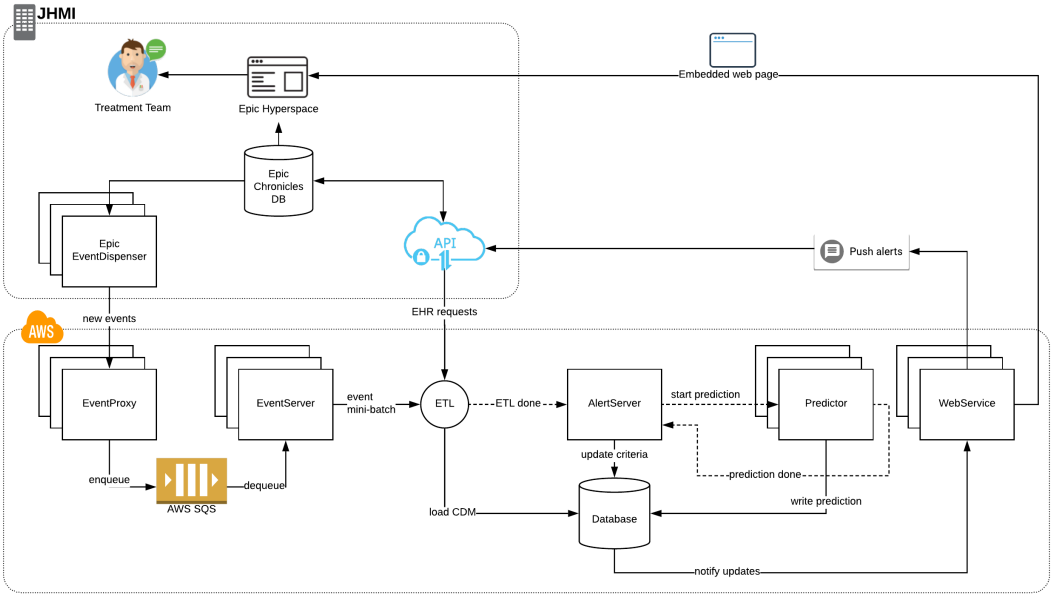


Figure 4.3: The implementation of the push-based online pipeline. It maintains always-on connections to receive EHR events instantly and trigger ETL mini-batchs to calculate real-time CDS.

database. After the ETL job is completed, an "ETL Done" message, which includes the enc_ids updated in this ETL job, is sent to the alert server. The alert server is an internal web service to coordinates the components in the online CDSS. Once it receives the "ETL Done" message, it forwards the message and distributes the prediction tasks to a series of connected predictors. The predictors will send a "Prediction Done" message back to the alert server after they make the predictions. Then, the alert server updates all CDS-related data models, e.g., criteria and alerts, and notify the web services with a "Invalidate Cache" message. Finally, the web services refresh their caches, update all the ongoing web sessions, and push alerts to the EHR vendor.

EHR system

In contrast, the push-based online data pipeline, as shown in Fig. 4.3, is triggered based on the events generated from the targeted EHR system. In this implementation, a group of scalable components — `EventProxy` — maintain the connections with the EHR vendor's event dispenser and forward new events to a buffer in the AWS Simple Queue Service (SQS). The SQS provides a queue which offers maximum throughput, best-effort ordering, and at-least-once delivery. Using the SQS guarantees all events can be buffered during an event burst. Another set of scalable components — `EventServer` — keep dequeuing the SQS buffer and run an ETL engine to execute such a mini-batch ETL job. Then, the rest of the pipeline is the same as the pull-based one.

The ETL mini-batch is implemented into two tasks. First, it dequeues all events from SQS every 15 seconds, and then parse the events to a list of web requests to run. The transformed results are saved as a CDM buffer; Second, it runs a loading task every 30 seconds to save all data from current CDM buffer to the online database. These two types of tasks are triggered by the timers independently and adaptively. Taking the loading task as an example, it starts to run every 30 seconds when the CDM buffer is not empty and the previous loading task has completed. Since the ETL engine uses `Asyncio` instead of multi-threading, we don't need to worry about race condition which makes the implementation much easier. Note that the mini-batch still needs to run web requests because the event payload does not contain the values updated in the EHR system, e.g., an "add flowsheet" event (e.g., add a new

heart rate) only contains the patient ID and the flowsheet ID but does not contain the value (i.e., heart rate and the timestamp).

In sum, the major benefits of using the push-based protocol, instead of pull-based one, are two-fold: First, the online pipeline can start to calculate new CDS instantly once the new events have been captured by the `EventServer`; while the pull-based one needs to wait for the trigger from the next timer event. Second, the push-based pipeline knows which patients have events generated and can only update on these patients, so it will be much more efficient because the amount of the latest updated patients should be much smaller than the number of all the bedded patients.

4.3 Evaluation

We evaluate the implementations of the pull and push-based protocols in our JHMI deployment in this section. The evaluation attempts to compare the cost and performance of the two protocols. Specifically, we evaluate the overhead of the online ETL on the targeted EHR system, e.g., the rate of online EHR requests, and the performance of the online service, e.g., the latency of the end-to-end ETL pipeline, for each protocol.

4.3.1 Experiment setup

We setup two separate pipelines with the different ETL protocols to compare the performance of the pull and push-based ETL pipelines, the infrastructure is list in Table 4.1. Also, the online EHR data source is served as REST APIs by the JHMI

Component	Usage	Instance	vCPU	Memory (GiB)	Replicas
Database	pull&push	db.m4.xlarge	4	16	1
`AlertServer`	pull&push	t2.medium	0.25	2	1
`Predictor`	pull&push	c4.large	2	3.75	1
`ETLJob`	pull	m4.large	0.25	1	3
`EventProxy`	push	r4.large	0.1	0.25	4
`EventServer`	push	r4.large	0.1	0.25	4

Table 4.1: The infrastructure used in the pull and push-based pipelines.

web service team. On one side, the three pull-based ETL jobs periodically request recent EHRs for bedded patients every 15 minutes, from the three hospitals, JHH, HCGH, and BMC, respectively.

On the other side, the four `EventProxy` instances guarantee that all incoming events can be pushed to the AWS SQS queue instantly. And the four ETL instances, i.e., `EventServer`, keep dequeuing all pending events [2] from the SQS buffer as a mini-batch with a minimum interval of 30 seconds. The four replicas can guarantee the SQS buffer never queue up in current experiment so that we can assume once the event is enqueue, it can be dequeue to a mini-batch immediately.

[2]The events included all five hospitals in JHMI since the event dispatcher pushed events from all hospitals in JHMI and no hospital filter exist so far

4.3.2 The online EHR request rates

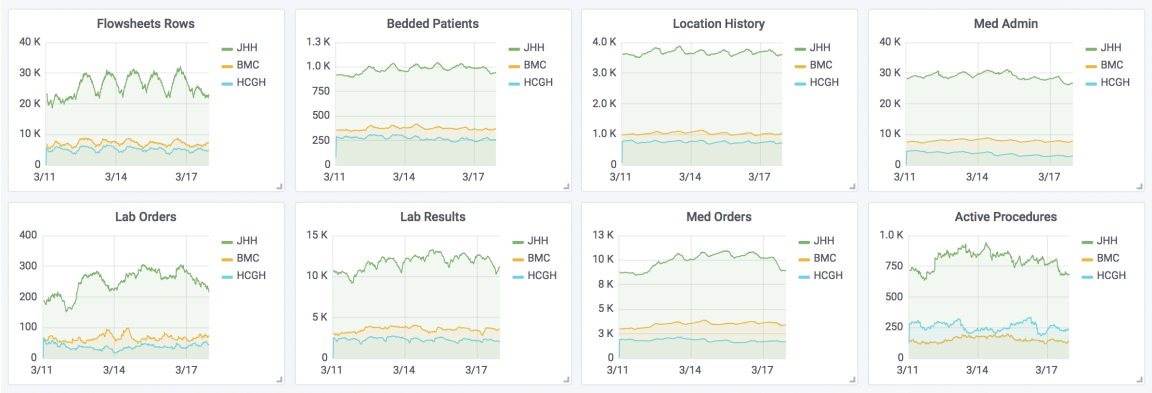
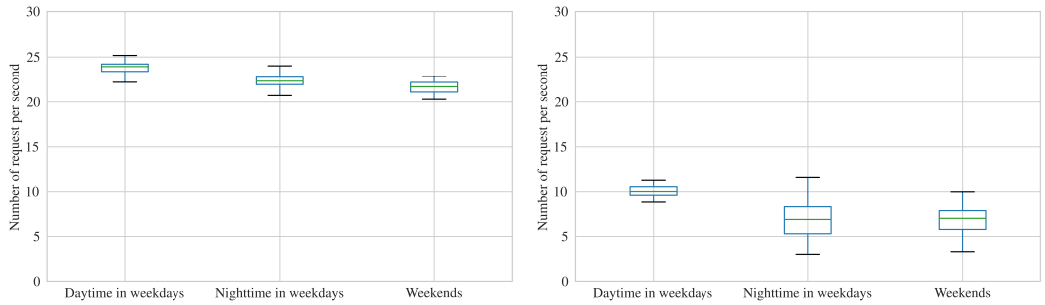


Figure 4.4: The number of EHR API requests per pull-based ETL over one week.

The online EHR request rate quantifies the major overhead of the online ETL associated with the targeted EHR system. Fig. 4.4 shows the number of API requests per pull-based ETL over one week. Taking JHH as an example, the number of bedded patients stabilized around one thousand. The two major types of requests were requests of flowsheet rows and medication administration. The number of requests for flowsheet rows in each pull-based ETL was floating between 20 to 30 thousand and reached to the peak during daytime in weekdays. Overall, the pull-based ETL needed more than 20 requests per second as shown in Fig. 4.5a.

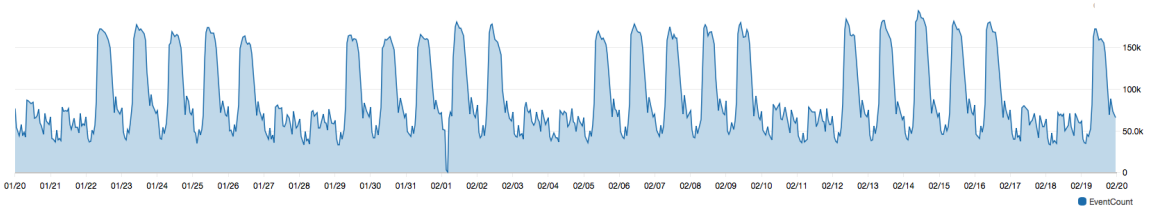
In contrast, the push-based ETL was triggered by the online EHR events. Fig. 4.6 shows the total number of events received over one month. Those events included both inpatient and outpatient events within all five hospitals in JHMI. The amount of events followed a weekly pattern, i.e., high volume in weekdays and low volume in weekends. It also followed a daily pattern in weekdays which indicated the hospitals



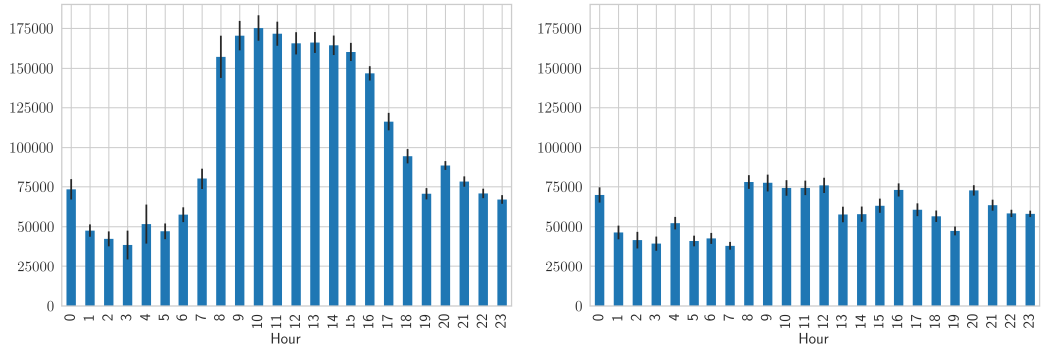
(a) Pull-based ETL **(b)** Push-based ETL

Figure 4.5: The rate of EHR web requests: pull vs. push

stayed in the peak time from 8 AM to 6 PM. Also, the peak time only occurred in weekdays.



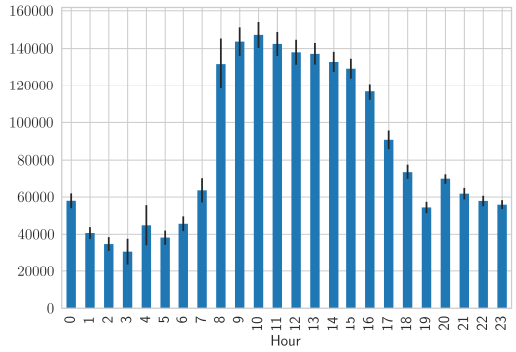
(a) Events per hour over one month



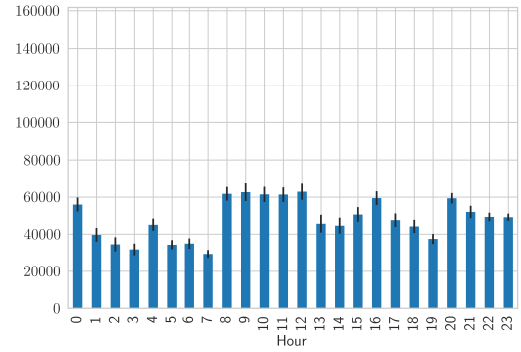
(b) Events per hour for weekdays **(c)** Events per hour for weekends

Figure 4.6: The total number of events received from JHMI EHR

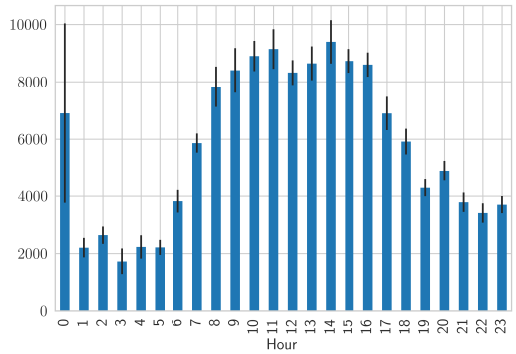
The statistics of the data events can be found in Fig. 4.7 and Fig. 4.8. Over 80% of data events were flowsheet related. Also, we can see that most of the medications



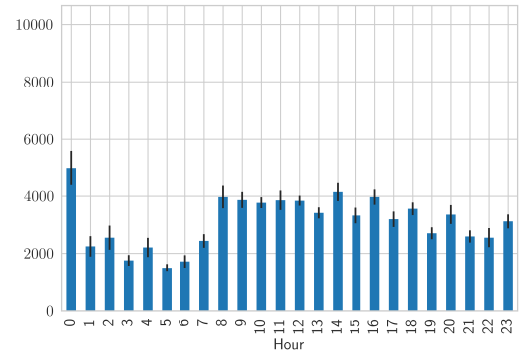
(a) Flowsheet added events per hour for weekdays



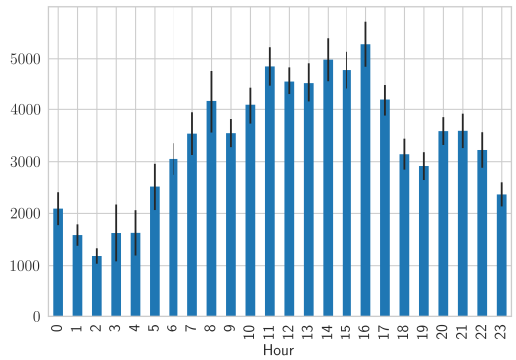
(b) Flowsheet add events per hour for weekends



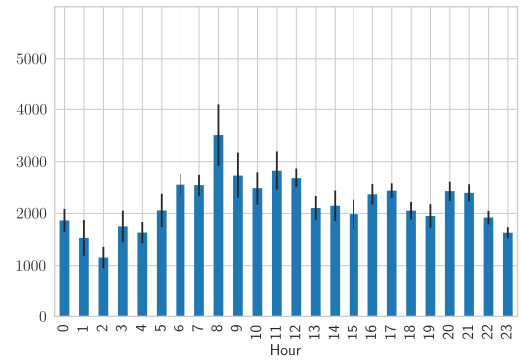
(c) Order signed events per hour for weekdays



(d) Order signed events per hour for weekends

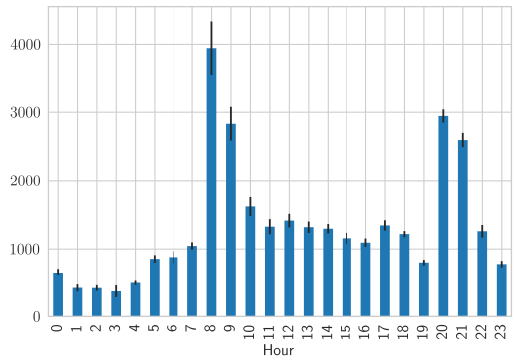


(e) Lab result updated events per hour for weekdays

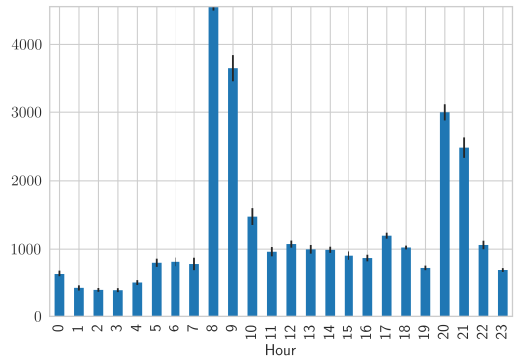


(f) Lab result updated events per hour for weekends

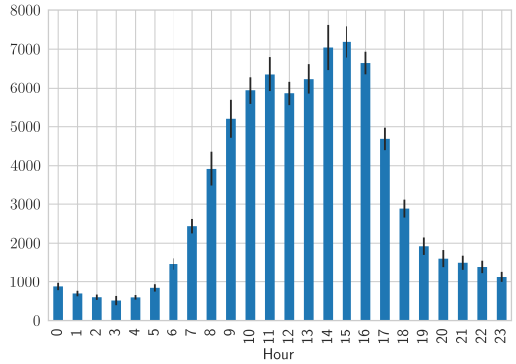
Figure 4.7: The key data events received from JHMI EHR (Part 1). Over 80% of data events are flowsheet row related.



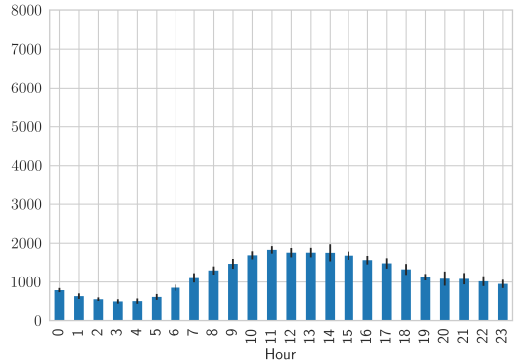
(a) MAR given events per hour for weekdays



(b) MAR given events per hour for weekends

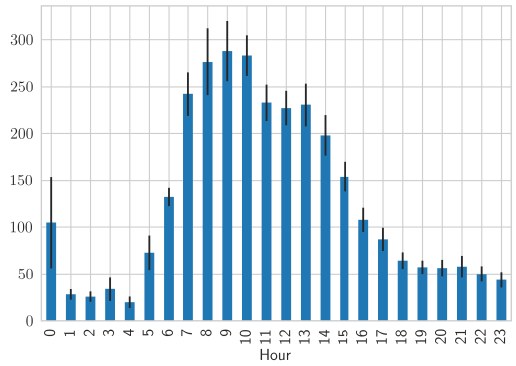


(c) Note updated events per hour for weekdays

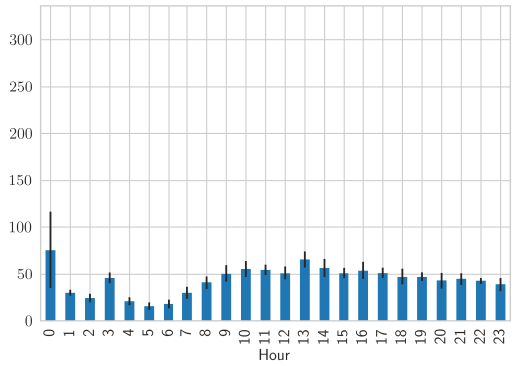


(d) Note updated events per hour for weekends

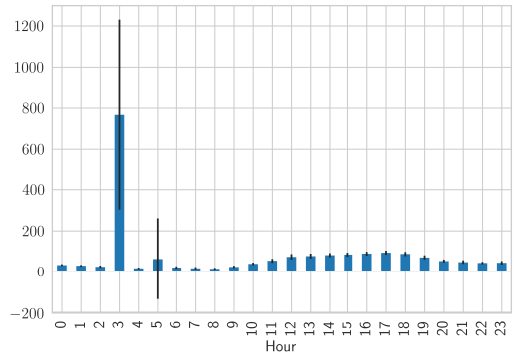
Figure 4.8: The key data events received from JHMI EHR (Part 2).



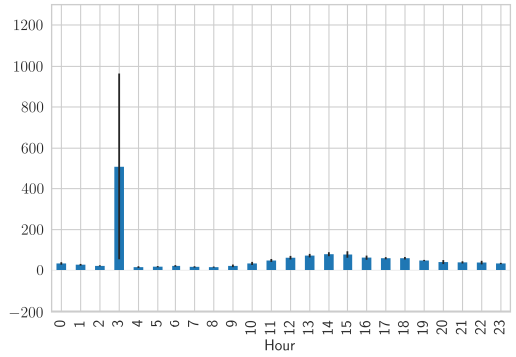
(a) Admit events per hour for weekdays



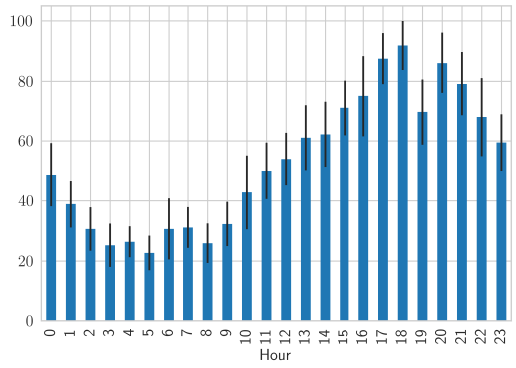
(b) Admit events per hour for weekends



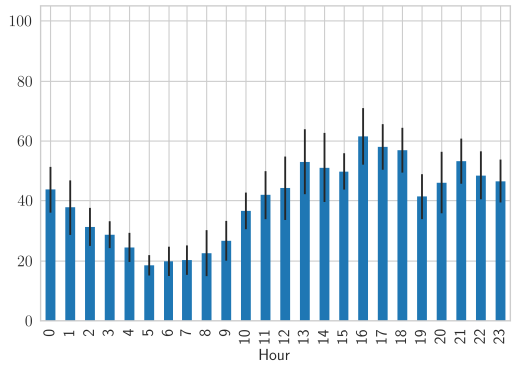
(c) Transfer events per hour for weekdays



(d) Transfer events per hour for weekends



(e) Discharge events per hour for weekdays



(f) Discharge events per hour for weekends

Figure 4.9: The ADT events received from the JHMI EHR.

were given from 8 AM to 9 AM and from 8 PM to 9 PM.

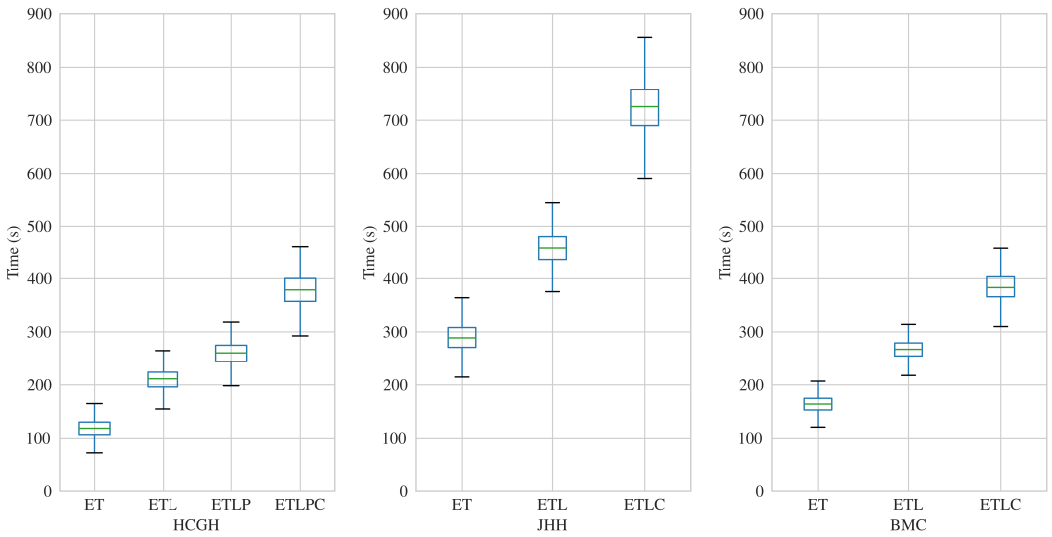
The ADT (admit, discharge, transfer) events are shown in Fig. 4.9. The peak of patient admission happened in the morning of weekdays, while, the peak of patient discharge happened around 6 PM. Some spikes in the figures, e.g., admission at midnight and transfer at 3pm were system generated events.

The push-based ETL consumed those EHR events and parsed them into web requests for current inpatients only. The rate of web requests by the push-based ETL was around 10 and 7 requests per second at the peak time (i.e., daytime in weekdays) and at the rest of the time respectively, as shown in Fig. 4.5b, which is less than half of the usage in the pull-based ETL.

4.3.3 The end-to-end latencies: pull vs. push

Another key benefit of using push-based protocol is the reduction of the end-to-end latency compared with the pull-based approach. We evaluate the end-to-end latencies of the pull and push-based online data pipelines here. The end-to-end latency measures the duration from the beginning of one ETL job to the completion of the TREWS prediction (short for P) and criteria calculations (short for C). Note that in our pilot study, only inpatients from HCGH has been predicted by TREWS, other patients bypassed the online prediction step.

Fig. 4.10 shows the latencies of the three hospitals using the pull-based ETL. The maximum latency was near 900 seconds for JHH (1,091 beds in total), i.e., it

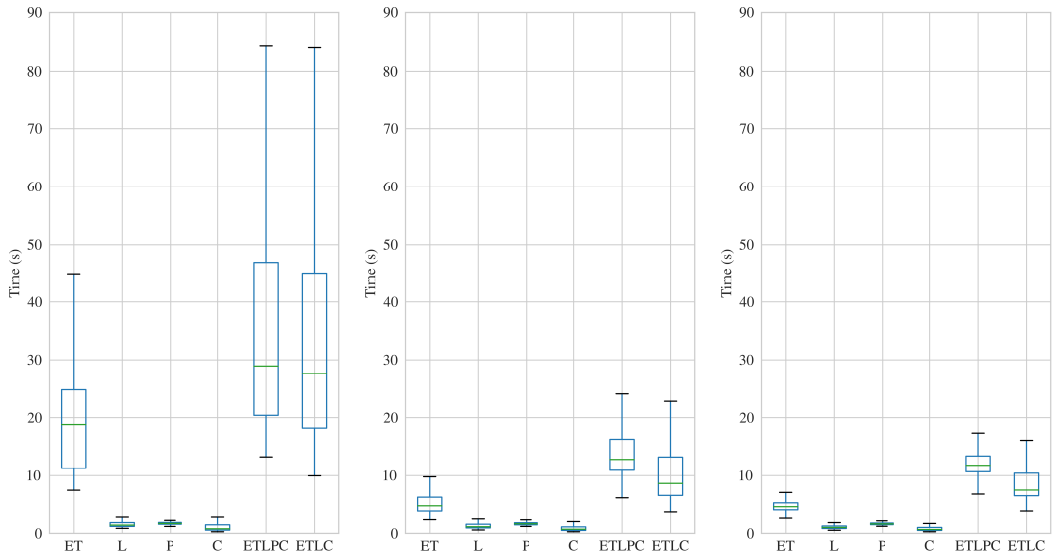


(a) HCGH (267 total beds) **(b)** JHH (1,091 total beds) **(c)** BMC (493 total beds)

Figure 4.10: The end-to-end latencies of the pull-based online data pipelines for the biggest three hospitals in JHMI. The average throughput of the full pull-based pipeline (HCGH) was 0.7 beds/s.

took near 15 minutes to extract EHR, transform to CDM, and then present them as CMS criteria in the web services. The ETL of HCGH needed near 480 seconds, i.e., 8 minutes, to complete the full pipeline including TREWS online prediction. The average throughput of the full pull-based pipeline was 0.7 beds/s.

Instead, Fig. 4.11 shows the end-to-end latencies using push-based ETL on three different time ranges, which are daytime in weekdays (peak time), nighttime in weekdays, and weekends. It is obvious the highest latency happened at the daytime in weekdays but all end-to-end latencies are within 90 seconds. The ET, L, P, and C shows the individual durations in each step respectively. ETLPC shows the end-to-end latency for HCGH patients who run full end-to-end pipeline including TREWS online prediction, while ETLC shows the end-to-end latency excluding online pre-



(a) Daytime in weekdays **(b)** Nighttime in weekdays **(c)** Weekends

Figure 4.11: The push-based end-to-end latencies (2,399 total beds). The average throughput of the end-to-end push-based pipeline (including prediction) was 80.0 beds/s.

diction for patients in JHH and BMC. Notice that the latency of ETLPC is higher than the sum of all individual components because it includes the gap between the two tasks in the ETL mini-batch, i.e., 1) parse and pull EHRs from the API and 2) transform EHRs into CDM. These two tasks ran independently and the results from the first task stored in a buffer for the second task to consume. The gap showed the interval when the first task was done but the second task was not ready to process the new results, i.e., the second task was running for the previous results. In sum, the average throughput of the end-to-end push-based pipeline was 80.0 beds/s.

4.4 Discussion

The CDS-Stack introduced both pull and push-based pipelines to deliver online clinical decision support. Especially for the push-based one, its maximum end-to-end latency is within 90 seconds, 10 times less than the latency using the pull-based one and the average throughput can achieve 80 beds/s, more than 100 times faster than the push-based one; furthermore, it only needs less than half of the requests to the API compared to the pull-based one. To our best knowledge, it is the first time to enable push-based online data pipeline in ML-based CDSS.

As the first step towards online ML-based CDS, it still has several limitations. One of the major limitations of the push-based data pipeline is that it is still not a complete push-based protocol because it still needs to request the REST APIs (i.e., pull data related to the latest events). In a complete push-based protocol, the event payload should include the updated data so that the ETL does not need to request the REST APIs again. An option to implement it is to use HL7 and FHIR protocols because it pushes the complete EHRs to the client, so the client does not need to pull again. The challenge is that HL7 and FHIR may not be available in some medical centers and customized EHRs may not deliverable in these protocols.

Another limitation is that the evaluation only focuses on the end-to-end performance inside CDS-Stack, specifically the ETL pipelines. A complete end-to-end performance evaluation should include the targeted EHR system as well. For example, to measure the latency from when an EHR item is typed in the system to when the CDS

is displayed to the treatment team. Such a complete end-to-end performance can give a picture of the entire closed loop. In sum, CDS-Stack provides a complete solution to serve both historical and online EHRs in CDM format for machine learning usage. We believe this solution can be used not just for inpatients or hospital settings, but also for outpatients or other third-party applications as well. For example, health data collected outside of the hospital can be transformed into CDS-Stack if a proper API is provided. For example, daily motion data, heart rate collected by wearable devices, can be imported from their APIs[36,37] to CDS-Stack by implementing the feature mapping for them. And the CDS-Stack, especially the CDM format and the pull and push-based pipelines, is general and simple to develop and apply machine learning-based CDS using its cloud infrastructure.

Chapter 5

The Mobile Parkinson Disease Score: Using Smartphones and Machine Learning to Quantify Parkinson Disease Severity

"We need to bring the exam room to where the patients are."

– Dr. Jay Sanders, telemedicine pioneer

In this chapter, our research moves from inpatient clinical decision support to outpatients, specifically patients living with Parkinson disease (PD). This chapter demonstrates how machine learning and smartphone sensing can be integrated to reform the monitoring and management of long-term illness like PD on a daily basis.

Current Parkinson disease measures are subjective, rater-dependent, and require in-clinic assessment.[38,39] As a result, clinical trials using these measures are long, expensive, and can miss or generate false signals.[39,40] Many PD motor symptoms are well-suited to objective measurement by smartphones.[41–43] Smartphone assessment has been evaluated in PD, but most studies focus on one specific feature (e.g. gait), rather than overall symptom burden.[43,44] We developed an Android smartphone application ("HopkinsPD") that assesses five activities (voice, finger tapping, gait, balance, reaction time) (See Appendix A),[45] which can be completed as often as desired, and enables reporting of medication administration. We create a mobile Parkinson Disease score (mPDS) to serve as an objective measure of PD, and test construct validity by evaluating 1) the ability of the mPDS to detect intraday symptom fluctuations; 2) the correlation between the mPDS and standard measures; and 3) the ability of the mPDS to respond to dopaminergic therapy.

5.1 Methods

5.1.1 Study population

Individuals with PD and Android smartphones were invited to download HopkinsPD through the Parkinson's Voice Initiative;[46] participants provided electronic consent for data analysis with application download. Data from participants who completed at least one complete set of activities before and after their first daily dose

of dopaminergic medication were used to develop the mPDS (development cohort). We also recruited and obtained written consent from individuals with and without PD to complete smartphone activities alongside traditional assessments, including the Movement Disorder Society Unified Parkinson's Disease Rating Scale (MDS-UPDRS), Hoehn & Yahr, and Timed Up and Go at baseline, month 3, and month 6 (clinic cohort). All study procedures were approved by the University of Rochester Research Subjects Review Board.

5.1.2 Creating the mPDS

Data from the development cohort were processed to extract novel disease features from each of the five activities (e.g. inter-tap interval from the finger tapping activity; See Appendix A).[45] Rather than replicating an existing PD score using regression, we used a rank-based machine-learning algorithm—disease severity score learning (DSSL)[47]— to derive an independent measure of PD symptom severity: the mPDS, which is scaled from 0-100 with high numbers reflecting high symptom severity. To weigh unique features, the algorithm exploits weak supervision[48] based on the assumption that symptom severity is higher at one time, immediately preceding dopaminergic medication administration, compared to another, one hour following medication administration. Given many such pairs, DSSL estimates a score by optimizing an objective function to correctly rank as many pairs as possible. Further description of the method can be found in Appendix B. Code for feature extraction and the DSSL

learning algorithm was open sourced at `https://github.com/dashan-emr/mpds`.

5.1.3 Ability of mPDS to detect intraday fluctuations

We evaluated the ability of the mPDS to capture symptom variability by evaluating the average intraday range in mPDS among home-performed assessments in those with PD in the clinic cohort.

5.1.4 Comparison of mPDS to traditional measures

Smartphone and traditional assessments completed within two hours of each other were used to compare the mPDS to traditional measures in individuals with PD. Pearson's correlation was calculated between the mPDS and the MDS-UPDRS III and total, Timed Up and Go, and Hoehn and Yahr.

5.1.5 Responsiveness of mPDS to dopaminergic therapy

We evaluated the ability of the mPDS to respond to dopaminergic therapy in the clinic cohort by comparing the mPDS derived during optional clinic-performed off- vs on-medication evaluations. A one-tailed Wilcoxon signed-rank test was used to

assess significance ($\alpha = 0.05$).

5.2 Results

250 individuals with PD downloaded HopkinsPD; 129 fulfilled requirements for the development cohort. 23 individuals with PD and 17 without PD constituted the clinic cohort. Baseline characteristics are shown in Table 5.1. Those with PD completed 58 in-clinic assessments (22 at baseline, 18 at month 3, 18 at month 6); those without PD completed 37 assessments (17 at baseline, 8 at month 3, 12 at month 6).

5.2.1 Creating the mPDS

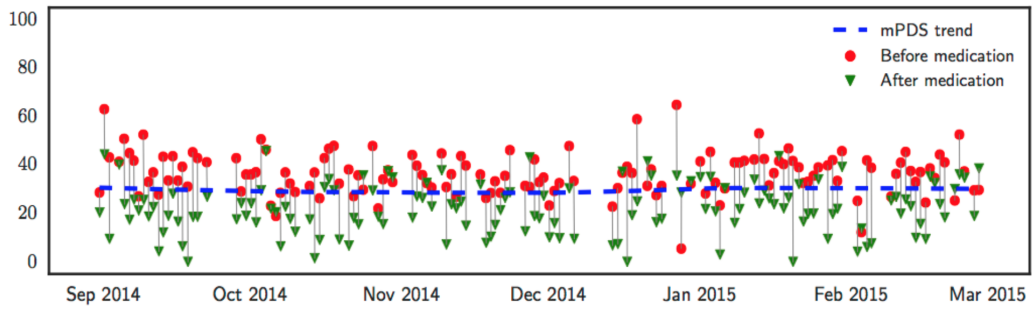
During 6 months, development cohort participants performed a mean (SD) of 48 (61) complete activity sets (range 2-278). 435 unique features were extracted from the five smartphone tasks. Eight features from the finger tapping activity, three from the balance activity, three from the gait activity, and one from the voice activity contributed most toward mPDS generation (Supplement). The relative weighting of features in generating the mPDS was gait (33.4%), balance (23.2%), finger tapping (23.0%), voice (17.0%), and reaction time (3.4%). The average mPDS (over all assessments) was 47% lower in controls (30.3, SD 15.0) than in those with PD (57.5, SD 16.9).

Characteristic	Smartphone users n = 250	Development cohort n = 129	Clinic cohort with Parkinson disease n = 23	Clinic cohort without Parkinson disease n = 17
Demographics				
Age (years)	57.2 (9.4)	58.7 (8.6)	64.6 (11.5)	54.2 (16.5)
Sex (% women)	95 (38.0%)	55 (42.6%)	11 (48%)	12 (71%)
Race (% white)	225 (90.0%)	123 (95.3%)	22 (96%)	16 (94%)
Ethnicity (% Hispanic/ Latino)	15 (6.0%)	9 (7.0%)	0 (0%)	0 (0%)
Education (% college graduate)	238 (95.2%)	121 (93.7%)	14 (61%)	8 (47%)
Using the internet or email at home (%)	250 (100%)	129 (100%)	21 (91%)	17 (100%)
Clinical characteristics				
Time since diagnosis (years)	4.4 (4.9)	4.3 (4.4)	7.0 (4.1)	N/A
Proportion taking levodopa (%)	96	97	90	N/A
MDS-UPDRS, total (I-IV) score	N/A	N/A	55.0 (26.5)	4.6 (4.6)
MDS-UPDRS III score	N/A	N/A	26.9 (11.2)	1.2 (1.7)
Timed Up and Go Test (seconds)	N/A	N/A	11.2 (3.3)	8.1 (1.3)
Hoehn & Yahr	N/A	N/A	2.1 (0.7)	0.0 (0.0)
We list mean (standard deviation) pairs except where indicated MDS-UPDRS = Movement Disorder Society —Unified Parkinson"s Disease Rating Scale				

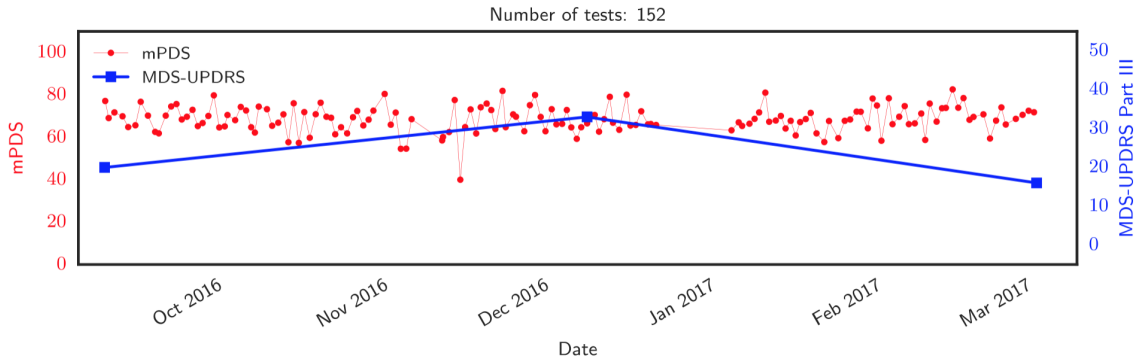
Table 5.1: Baseline characteristics.

5.2.2 Ability of mPDS to detect intraday fluctuations

During 6 months, clinic cohort participants performed a mean (SD) of 210 (323) complete activity sets (range 2-996). The mPDS detected an average intraday change of 13.9 (SD 10.3) among those with PD. Fig. 5.1a depicts intraday severity fluctuations. The average MDS-UPDRS IV score was 4.6 (SD 4.3).



(a) Intraday severity fluctuations captured for one participant



(b) Assessment frequency of mPDS vs. Movement Disorder Society-Unified Parkinson"s Disease Rating Scale Part III over six months

Figure 5.1: Mobile Parkinson disease score (mPDS) assessment scores captured over 6 months

5.2.3 Comparison of mPDS to traditional measures

Sixteen smartphone and traditional assessment pairs met criteria for analysis. Table 5.2 shows the correlation matrix between the mPDS and traditional measures, demonstrating good to excellent correlation. Fig. 5.1b demonstrates the ability of the mPDS to monitor symptom severity more frequently than traditional measures.

Table 5.2: Correlation matrix between the mobile Parkinson disease score (mPDS) and traditional Parkinson disease outcome measures (n=16).

	mPDS[a]	MDS-UPDRS[b] III	MDS-UPDRS Total	Timed Up and Go Time	Hoehn & Yahr Stage
mPDS	1.00				
MDS-UPDRS III	0.88	1.00			
MDS-UPDRS Total	0.81	0.82	1.00		
Timed Up and Go Time	0.72	0.74	0.27	1.00	
Hoehn & Yahr Stage	0.91	0.96	0.80	0.70	1.00

[a] mobile Parkinson disease score [b] Movement Disorder Society-Unified Parkinson"s Disease Rating Scale

5.2.4 Responsiveness of mPDS to dopaminergic therapy

Seven off- and on-medication assessment pairs were performed in the clinic cohort. The mPDS decreased by an average of 16.3 (SD 5.6) in response to dopaminergic therapy with significant Wilcoxon signed-rank test (W=28, p=0.008). The MDS-

UPDRS III decreased by an average of 10.4 (SD 4.6) in response to dopaminergic therapy.

5.3 Discussion

The mPDS is a novel measure that provides rapid, remote, frequent, and objective assessment of PD symptom severity on widely available smartphones. We demonstrated construct validity by showing that the mPDS can: 1) capture intraday fluctuations characteristic of PD; 2) correlate with traditional PD measures; and 3) respond to dopaminergic medication.

The mPDS is complementary to traditional PD measures. First, assessments can be performed frequently in real-world settings.[49] Second, the score provides an objective measure of PD symptom severity, not impacted by inter-rater variability.[50] Third, the mPDS, unlike traditional measures, objectively weighs activity features. The MDS-UPDRS III is biased toward tremor-predominant disease1 with only five of 33 items assessing gait or balance. In contrast, 56.6% of the mPDS is derived from gait or balance activities. Last, unlike traditional measures, which take years and significant resources to develop,1 the mPDS was generated quickly, from a relatively small number of participants using automated techniques that can account for noise in data collected from multiple smartphone sensors and self-report of medication administration.[51] Combining smartphone data with the machine-learning methods outlined

here may also provide opportunities for developing objective severity measures in other neurological conditions.

This study has several limitations. Participants were generally white, college-educated, Android smartphone owners, and not representative of the broader PD population. Only 52% of those who downloaded the application met criteria for inclusion in the development cohort. Additionally, the clinic cohort included only seven assessments to evaluate the responsiveness of the mPDS to dopaminergic therapy, and only 16 smartphone and in-person assessment pairs met criteria for the correlation analysis. Still, this represents one of the largest longitudinal smartphone assessment of PD.

Further validation of the mPDS in a larger sample with patient-relevant anchors is needed. New iterations of the application for Android and iOS smartphones will expand participation and include additional features and functionality that could provide new insights into PD.

Chapter 6

Related Work

"To learn without thinking is blindness; to think without learning is idleness."

– Confucius, *The Analects of Confucius*

This dissertation research stands on the shoulder of prior work in a number of ways. This chapter describes previous work, how it has inspired this dissertation, and the contributions that this dissertation offers beyond existing research. Section 6.1 discusses the current infrastructure which enables machine learning in both academy and industry. Section 6.2 focuses on the deployment and practice of using machine learning in clinical decision support. Section 6.3 discusses the current open EHR data models. Section 6.4 discusses state of the art in using machine learning and mobile technologies to track Parkinson disease in the wild.

6.1 Infrastructure for Machine Learning Practice

6.1.1 Motivation

Machine learning is being deployed in a growing number of applications. Over the past five years, voice-driven personal assistants have become commonplace, image recognition systems have reached human quality, and autonomous vehicles are rapidly becoming a reality.[52] A series of open-source machine learning frameworks are available now to accelerate the core development of machine learning, i.e., training and prediction. Those frameworks includes Scikit-learn,[17] Apache Spark MLLib,[18] TensorFlow,[19] Caffe,[53] and Torch,[54] etc. However, those frameworks mainly contribute to model training or evaluation but not deployment and maintenance. In practice, the core of machine learning — training or prediction — is a comparatively small, albeit critical, part of the overall application lifecycle.

Sculley et al.[13,55,56] sought to increase the community's awareness of the severe technical debt that must be considered and paid in practice over the long term for machine learning systems. They explored several ML-specific risk factors to account for in system design. These included boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns. They hoped to encourage

additional development in the areas of maintainable ML, including better abstractions, testing methodologies, and design patterns. They argued both engineers and researchers should be aware of this issue since research solutions that provide a small accuracy benefit at the cost of massive increases in system complexity are rarely wise practice. Even the addition of one or two seemingly innocuous data dependencies can slow further progress.

Lin et al.[14] also shared the Twitter experience in scaling big data mining infrastructure. The most prominent lesson they learned was that successful big data mining was about much more than what most academics would consider data mining. A significant amount of tooling and infrastructure was required to operationalize vague strategic directives into concrete, solvable problems with clearly-defined metrics of success. The infrastructure and system design should help data scientists (who spent a significant amount of effort performing exploratory data analysis even to figure out "what's there", including data cleaning and data munging not directly related to the problem at hand) and data infrastructure engineers (who worked to make sure that productionized workflows operate smoothly, efficiently, and robustly, reporting errors and alerting responsible parties as necessary).

6.1.2 Practices

Since building machine learning applications remains prohibitively time-consuming and expensive for all but the best-trained, best-funded engineering organizations, the

Stanford DAWN project[52] aimed to provide a stack of systems and tools to support usable, end-to-end machine learning applications. The ultimate and ambitious goal of this on-going project was to democratizing machine learning technology and let one or two domain experts can build their production-quality data products (without requiring a team of PhDs in machine learning, big data, or distributed systems, and without understanding the latest hardware). DAWN attempted to achieve their goal from three directions: One, to develop new interfaces from model specification to model monitoring to empower domain experts who are not ML experts. For example, they have obtained promising early results with a new system called Snorkel[57] that produces high-quality models from low-quality rules. Also, Rong et al.[58] utilized automatic smoothing to improve streaming time series visualization and human attention. Two, to design end-to-end systems that encapsulate the whole ML workflow and hide internals from users, similar to a search engine or a SQL database. One example was MacroBase,[59] an analytics engine and architecture for analyzing fast data streams, which combined streaming outlier detection and streaming data explanation. Three, to develop new computational substrates, from language support to distributed runtimes and accelerated hardware, so that training and deploying ML can be quick and in a cost-effective manner. For instance, they were developing Weld,[60] a new runtime that can optimize data-intensive code across different libraries and functions to automatically generate fast implementations either for ML training or serving. Weld can already accelerate modern data analysis tools such as Apache

Spark, Pandas and TensorFlow by 10 times by optimizing across the operators within them, and can accelerate cross-library workloads by up to 30 times.

Clipper[61] was a general-purpose low-latency prediction serving system interposing between the end-user applications and a wide range of machine learning frameworks. It introduced caching, batching, and adaptive model selection techniques to reduce prediction latency and improves prediction throughput, accuracy, and robustness without modifying the underlying machine learning frameworks.

Other example of industry machine learning production system is Michelangelo[62] made by Uber. Uber faced several challenges of building and deploying machine learning models related to the size and scale of their operations. For example, those challenges included the tool separation between data scientists and engineering team, i.e., a ML model developed by data scientists needed to reimplement again to fit production environment by the engineering team; and no system in place to build reliable, uniform, and reproducible pipelines for creating and managing training and prediction data at scale. Michelangelo was designed to address these gaps by standardizing the workflows and tools across teams though an end-to-end system that enables users across the company to easily build and operate machine learning systems at scale.

6.1.3 Discussion

From our experience in developing CDS-Stack and deploying it into real healthcare practice, we faced the similar system and infrastructure challenges aforementioned

in above relate work. Especially, Sculley et al.[13] and Lin et al.[14] described and summarized the problems we were facing.

In our study, the first question we had was how to quickly and efficiently establish an end-to-end machine learning system on the cloud? Was there a software we can use to deploy and maintain the end-to-end CDSS and iterate the machine learning model? Unfortunately, there was no such an open-source software available. Even though the research articles aforementioned in Section 6.1.2 provides a series of promising systems and algorithms in this domain, they commonly abstracted a particular or partial functionality from a real end-to-end system and improved it with novel ideas. So it is impractical to apply those ideas without a baseline system has been set up.

These are the leading reasons we built and open sourced CDS-Stack and share with the peers who also aim to reform healthcare using machine learning. CDS-Stack can be used to establish a testbed or infrastructure for developers to experiment actionable end-to-end ML-based CDS and keep improving it from different aspects. For example, the idea of automatic smoothing[58] can be adopted in CDS-Stack and can be compared with baseline visualization; Also MacroBase[59] can be integrated to setup streaming outlier detection and streaming data explanation. One of the future work for CDS-Stack is to improve its modularity as a testbed to plug and play these new ideas, not just new machine learning models.

6.2 Clinical Decision Support System and Machine Learning

6.2.1 Clinical decision support system from a practical view

Despite promising initial studies from the academia, the majority of CDSS in practice have not provided features beyond general alerts, reminders, summary dashboards, and automated information retrieval systems.[63] A majority of United States hospitals have yet to implement any form of CDSS.[64,65] "Comprehensive" EHR systems are those that include decision support features (clinical guidelines, clinical reminders, drug-allergy alerts, drug-drug interaction alerts, drug laboratory interaction alerts, drug-dosing support); only 1.5% of 2952 hospitals surveyed achieved a "comprehensive" classification.[66]

Taking JHMI as an example, clinical decision support is embedded within Epic EHR system in a variety of formats. Specifically, the Epic Decision Support workgroup is primarily responsible for configuring alert-based decision support interventions including medication alerts, Best Practice Advisories (BPAs), and health maintenance. Even in a top medical institute like JHMI, only less than 20 clinical decision BPAs have been deployed and most of them are simple and rule-based screening tools and guidelines, e.g., BMI screening and follow-up plan,[67] diabetes nephropathy

Screening,[68] and Pneumonia Vaccination.[69]

Overall, few studies of CDSS have shown any improvement in outcomes, and any such effects seen have achieved only low statistical significance.[64] A meta-analysis of 148 randomized clinical trials on CDSS implementation found that only 20% influenced clinical outcomes.[65] Of these, improvements were seen in morbidity outcomes such as number of hospitalizations, surgical site infections, cardiovascular events, and deep vein thrombosis, but there was little effect on mortality or pharmacologic adverse events,[65] suggesting that CDSS will need improvement before they can routinely provide clinically meaningful knowledge.

6.2.2 Diagnostic machine learning algorithms using EHRs

Applying machine learning into clinical decision support becomes a hot topic and wildly believed is that it will improve the quality of care and even surpass human experts. Recently, more and more retrospective analysis studies show machine learning can be a useful tool in disease diagnosis, especially for some complex diseases,[70] clinical events, and procedures, e.g., sepsis,[7–9,71,72] heart failure,[73] cancer,[10] asthma, Parkinson disease,[11,41,45,74] medical images like chest X-ray,[75,76] etc. In those studies, historical EHRs have been extracted, labeled, and evaluated by specific machine learning algorithms with metrics like the area under the curve (AUC), positive pre-

dictive value (PPV), sensitivity and specificity, detection hours before onset, etc.

6.2.3 Systems designed for ML-based CDS

However, the systems for the deployment of online machine learning-based CDS are relatively rare. We discuss state of the art on the practices of ML-based CDS below.

Ng et al.[77] developed the PARAMO system, a predictive modeling platform which constructed a large number of pipelines in parallel with MapReduce/Hadoop. However, PARAMO was built on the user's own cluster, which was not always available in every clinical institution, and also lack scalability when faced with large datasets beyond the capacity of their existing cluster. Also, most pipelines such as PARAMO were challenging to deploy in a clinical setting due to the significant expenses required to maintain servers. Therefore, these systems made little to no impact on clinical decision-making.

To help address the limitations of PARAMO, Chen et al.[70] developed and deployed a hybrid system that combined a secure private server with the cloud-based Amazon Web Services (AWS) Elastic MapReduce platform. The system consisted of a web service that ran on a private server in a secure environment for preprocessing patient data into feature matrices, and an on-demand AWS web service to perform predictive modeling computations. Such a hybrid setup attempt to enable security of the patient data and at the same time leveraged the scalable computing infrastructure on the

cloud.

6.2.4 Discussion

The CDS-Stack is different from above two systems in the following sense: First, CDS-Stack is open and reusable, while the above two systems were not available outside of the lab. Second, CDS-Stack is a pure cloud-based system. It is more scalable and manageable than the private cluster, e.g., PARAMO. Third, CDS-Stack provides HIPAA secure and compliant implementation, so the hybrid design in [70] is not necessary. A secure and cloud-based system can directly integrate with the EHR vendor which does not need to set up a private cluster as a gateway. Last, CDS-Stack also includes the database layer and live end-to-end pipeline to deliver CDS and the above two systems still mainly focused on offline training.

6.3 EHR Data Models

6.3.1 Current EHR data models

Taking the industry leader, Epic, as an example, the current EHR is relative closed for the purpose of developing machine learning-based CDS. First, the database schema and data are closed, developers need to get trained and certified so that they can access the data schema and the data. There is no open document or playground

for developers to start learning. Second, the Epic system was implemented based on versions of the antiquated but powerful programming language (MUMPS). Therefore, there is no machine learning library and experts can directly work on the antiquated platform and build machine learning applications. Third, the interoperability is limited due to the Epic's software distribution model: it customizes its EHR software for each client which limits the interoperability between sites and the capacity to communicate with providers who use other EHRs. While there is potential for EHRs with functional data linkages to be used for research purposes, currently the lack of interoperability between EHRs hinders research capabilities.[78]

6.3.2 Open EHR data models

Currently, enabling the broader usage of EHRs can be done in two ways: restricted assess and altered data. The integrating Biology and Bedside (i2b2) project[79] allows researchers to query for aggregate summaries of the data without access to individual level information. Similar to CDS-Stack, i2b2 also provides a simple data schema that allows very diverse patient data from the enterprise to be integrated into a few central tables that simplify the search and analyze on the data that the enterprise makes available on its patients. Besides, another notable success in the release of data in critical care is the PhysioBank component of PhysioNet, and in particular the Multiparameter Intelligent Monitoring in Intensive Care II and III (MIMIC-II, MIMIC-III) database.[29,80] The data has been provided to researchers after certifi-

cation of completion of a human subjects training course and the signing of a data use agreement. The database is a great step toward removing barriers between researchers and real-world data necessary to validate their work. Both i2b2 and MIMIC databases de-identified PHI (patient health information) and provides anonymous patient data, the purpose is to share the data to external researchers. The two above architectures focus on EHR analysis and does not provide any support on applying the analytical results back to the EHR system.

6.3.3 Discussion

Therefore, instead of aiming at EHR data sharing, the goal of CDS-Stack is to provide this open source tool to help researchers or developers build their own ML-based CDSS. CDS-Stack provides a pipeline constructor to turn EHR into data frame so that researchers can focus on ML relevant development. Whiling using the CDS-Stack, researchers/developers can reuse the database schema and most of the ETL pipeline in both the development and production stage. However, ML models trained on i2b2 or MIMIC-II need to create the online data pipeline from scratch.

6.4 Machine Learning and Remote Monitoring for Parkinson Disease

Remote monitoring of Parkinson disease is needed to improve the quantity and quality of care for PD. Currently, individuals living with PD have access to care or participate in research primarily during in-person clinic or research visits, which take place at most once every few months. More frequent clinic visits are limited by travel distance, increasing disability, and uneven distribution of doctors.[81] Without remote monitoring, it is difficult for clinicians to provide optimal treatment for their patients based on these periodic "snapshots" of the disease progression alone.[82]

6.4.1 The readiness of sensing technology for Parkinson remote monitoring

Existing studies have required the use of specialized and expensive medical devices such as accelerometers, gyroscopes, EEG and passive infrared sensors.[83–89] Many of these studies have also only reported data collected in the laboratory setting,[83–85,88,89] which does not faithfully represent the patterns of variability that individuals with PD may experience at home. Also, the majority of past studies have focused on monitoring only one aspect of PD such as dyskinesia,[85] gait,[86,88,90–92] voice,[93] postural sway,[90] and resting tremor.[89,94] However, PD is a multi-faceted disease with many

varied symptoms. Thus, a multi-dimensional approach is needed to monitor all PD symptoms at home continuously.

6.4.2 Using smartphone and machine learning in remote monitoring of Parkinson disease

Smartphone-based tracking and measurement tools offer a promising new avenue for monitoring progressive symptoms outside the clinic. Moreover, without the need for expensive specialized medical hardware, new software tools can be easily downloaded and installed on an individual's smartphone for in-home monitoring. For example, Apple's ResearchKit[95] enabled several clinical studies to collect patients' data from smartphones remotely. The targeted population of those clinical studies included patients with Parkinson disease,[96] Asthma,[97] gait problems,[98] melanoma risk,[99] cognitive problems,[100] etc.

With the increasing data size and complexity, machine learning provides a solution for researchers/physicians to analyze the collected data and turn it into meaningful clinical related decision support. Taking Parkinson disease as an example, Several studies introduced machine learning to diagnose Parkinson disease, i.e., distinguish Parkinson patients from healthy controls.[41,45,90] And other studies focus on particular symptoms, e.g., tremor,[101,102] gait.[103] Those studies suggested a promising direction to use smartphone and machine learning to quantify the symptoms of Parkinson

disease.

6.4.3 Discussion

Most of the aforementioned PD studies using machine learning and smartphones still did the experiments in a lab setting on a relatively small number of participants. Instead, our study adopted a pure remote and at-home setting, similar to the studies using Apple's ResearchKit, and the number of participants exceeded other contemporaneous studies. Such a large and remote study also addressed a new crucial problem, i.e., how to get labels (i.e., Parkinson severity) for those remote participants? Without clinically validated labels, it is impossible to use supervised learning, the dominant machine learning approach in existing studies. Therefore, our study introduced a weak learning method and utilized the symptom changes before and after medication to learn a severity model for Parkinson. Meanwhile, our study provided a physician-understandable score, mPDS, which is similar to standard Parkinson assessment scores, to quantify severity. Also, mPDS is a unified score measuring multi-dimensional symptoms including voice, gait, balance, and dexterity, and provided a high-frequent and overall answer to the daily Parkinson severity variance which used to be hidden to physicians.

Chapter 7

Conclusion and Future Work

This dissertation has reported the journey we took to bring machine learning-based clinical decision support into clinical practices. During this journey, we have explored how to build cloud-based infrastructure to support the development, deployment, and maintenance of machine learning-based CDS from scratch. We have contributed the core of our implementation — CDS-Stack — the open-sourced platform to accelerate the process of building end-to-end ML-based clinical decision support systems for all ML practitioners in healthcare. On top of this platform, we have designed CDM, a common data model, to represent EHRs in a specific and straightforward format for ML usage. We also have implemented an ETL engine for transforming raw EHRs into CDM in parallel. Both pull-based and push-based data pipelines have been provided to enable live CDS delivery back to the EHR system. Using CDS-Stack, we have deployed a sepsis early warning alerts — TREWS — into JHMI hospitals and

support tens of thousands of inpatients since 2017. We believe CDS-Stack could be an invaluable tool to fill the gaps between machine learning and healthcare practice and catalyze the practice of using machine learning in healthcare.

This dissertation has also shown the benefits of extending ML-based CDS from in-hospital settings to daily health monitoring and management. An example application mPDS has been proposed to quantify Parkinson severity using smartphone and machine learning. It has demonstrated that the power of ML-based CDS is not just assisting diagnosis, but also generating new meaningful information in disease management.

Still, the contributions of this dissertation present the first step towards realizing machine learning-based clinical decision support. More work needs to be done to build a robust, extensible, and clinician-friendly clinical decision support ecosystem. This dissertation attempts to list several promising future directions as follow.

First, the dramatical explosion of health data streams is still on-going, including EHRs recorded the hospitals and other health-related, personal, physiological, psychological, and social data generated from mobile, wearable, and any other networked devices. Therefore, human attention-based or manual data exploration and data quality control seem increasingly impossible to satisfy the future clinical decision support system. We have to rely on automation to meet the challenge of this live, heterogeneous, and big data volume. Integration of automation system and outlier detection algorithms can be a potential solution for this purpose. One challenge is

how to build such a detector with high sensitivity and specificity to detect not just outliers but also distinguish errors with clinically meaningful information.

Second, CDS-Stack is available but need to further extend to other EHR systems and practices. We hope CDS-Stack can be an open testbed tool for other researchers to establish more machine learning-based clinical decision support system. New applications, machine learning models, and system ideas or tools can be evaluated on such a modular and scalable system.

Last but not least, with the catalyst of CDS-Stack, more practical applications, like mPDS, should be developed and evolved in healthcare practice. In summary, this dissertation hopes can share our experiences to flatten bumps in the road for those practitioners who come after us. Also, for academic researchers, it attempts to provide a broader context for machine learning-based clinical decision support in production-level and point out opportunities for future work.

Appendices

A HopkinsPD Android Application

HopkinsPD is an open source Android application to collect smartphone activities related to Parkinson disease, as shown in Fig. A.1 and Fig. A.2. HopkinsPD server provides basic web-based visualization tool for researchers, see Fig. A.3. As a research platform, HopkinsPD has enabled multiple research projects since it was deployed in 2014. [11,41,45,74,90,104–109] The source code of HopkinsPD is available at `https://github.com/zad/HopkinsPD_Android`.

Table A.1 list all implemented smartphone activities related to Parkinson disease.

Table A.2 provides an exhaustive list of the features extracted from the five types of activities, along with a brief description of each feature. Acceleration features were based on definitions used in previous studies.[74112113] Acceleration features were computed from the tri-axial acceleration time series (x, y, and z-axis), as well as the spherical transformation of the tri-axial acceleration time series (i.e., radial distance,

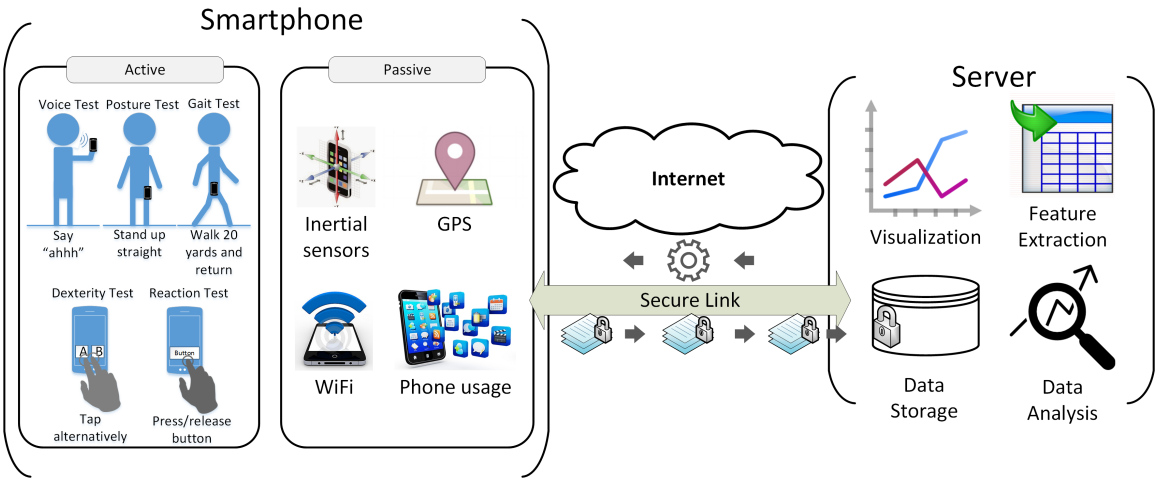
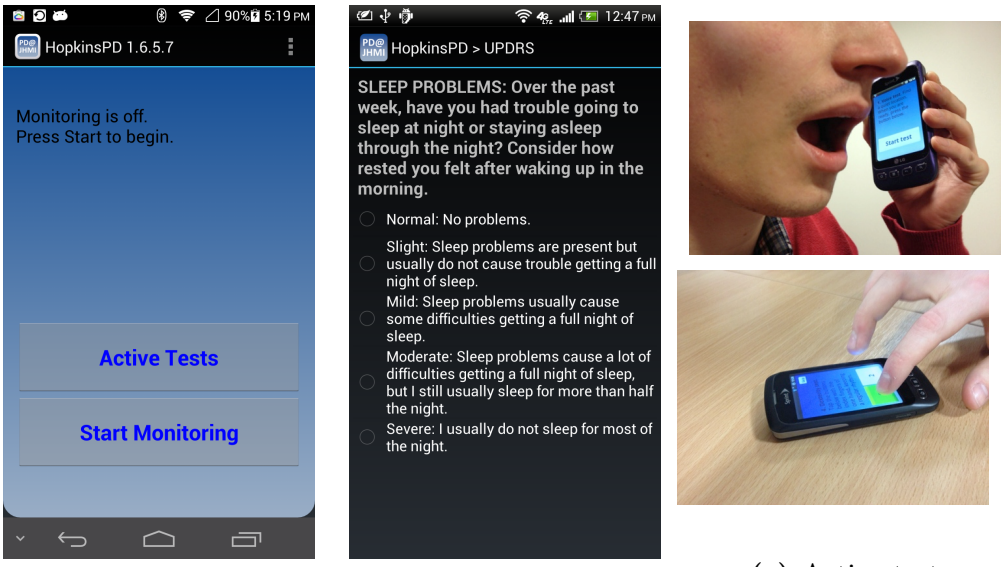
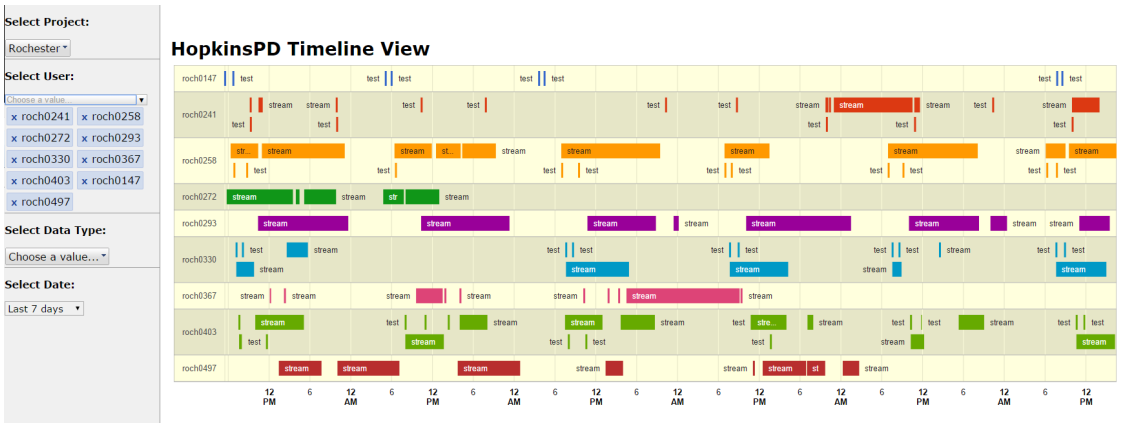


Figure A.1: HopkinsPD architecture.

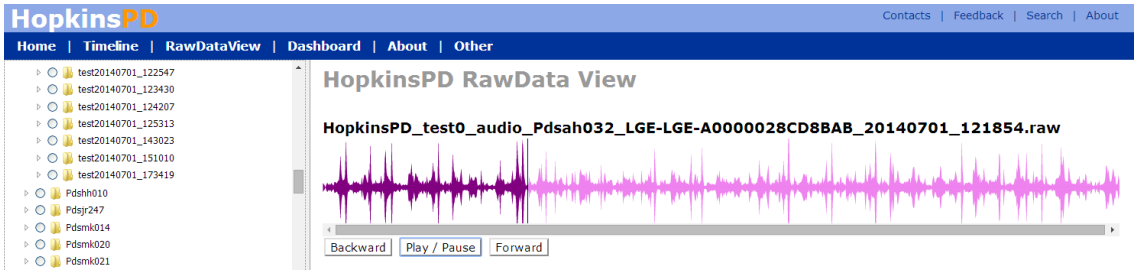


(a) Primary interface **(b)** Self-report **(c)** Active tests

Figure A.2: HopkinsPD mobile application.



(a) Timeline: user monitoring



(b) An example of test view: voice view

(c) Part of the day view

Figure A.3: HopkinsPD web-based visualization.

Table A.1: Description of smartphone activities in HopkinsPD

Activity	Description	Relevant PD symptoms	Related MDS-UPDRS III Motor exam	Instruction
Voice	Place the phone to your ear as if making a normal phone call, take a deep breath, and say "aaah" for as long and as steadily as you can.	Dysphonia	3.1 Speech	https://www.youtube.com/watch?v=BeUFgljiuJI
Balance	Stand up straight unaided and place the phone in your pocket for 30 seconds.	Postural instability	3.12 Postural stability 3.13 Posture	https://www.youtube.com/watch?v=xwJsLGdlhsE
Gait	Stand up and place the phone in your pocket. When the buzzer vibrates	Bradykinesia	3.8 Leg agility	https://www.youtube.com/watch?v=2BidIYn1Nrg
	walk forward for 20 yards; then turn around and walk back.	Freezing of gait	3.10 Gait, 3.11 Freezing of gait	
Dexterity	Place the phone on a surface such as a desk or table. Tap the buttons	Bradykinesia	3.4 Finger tapping	https://www.youtube.com/watch?v=tJLqvKHn2XQ
	alternately with the index and middle finger of one hand, keeping a regular rhythm.	Reduced dexterity	3.5 Hand movements	
Reaction	Keep the phone on a surface as before. Press and hold the on-screen button (i.e., at the bottom of the screen) as soon as it appears; release it as soon as it disappears.	Bradykinesia Reduced dexterity	3.5 Hand movements	https://www.youtube.com/watch?v=Brz2yZp_O7M
Rest Tremor*	Sit upright, hold the phone in the hand most affected by your tremor, and rest it lightly in your lap.	Resting tremor	3.17 Rest tremor amplitude	https://www.youtube.com/watch?v=cPd1Ct0xOCg
Postural Tremor*	Sit upright and hold the phone in the hand most affected by your tremor outstretched straight in front of you.	Postural Tremor	3.15 Postural tremor of the hands	https://www.youtube.com/watch?v=6QjjBa1HFVk

* Activities have been implemented but not used in mPDS.

Table A.2: Brief description of features extracted for active tests

Feature	Brief Description
Acceleration Features[a]	*mean*: Mean *std*: Standard deviation *Q1*: 25^{th} percentile *Q3*: 75^{th} percentile *IQR*: Inter-quartile range (IQR) ($Q3 - Q1$) *median*: Median *mode*: Mode (the most frequent value) *range*: Data range (max - min) *skew*: Skewness *kurt*: Kurtosis *MSE*: Mean squared energy *En*: Entropy *MCR*: Mean cross rate *DFC*: Dominant frequency component *AMP*: Amplitude of DFC *meanTKEO*: Instantaneous changes in energy due to body motion[b] *AR1*: Autoregression coefficient at time lag 1 *DFA*: Detrended fluctuation analysis[110] *XCORR*: Cross-correlation between two axes *MI*: Mutural information between two axes *xEn*: Cross-entropy between two axes
Voice Features	*Len*: Voice duration in seconds *AMP*: Voice amplitude *F0*: Dominant voice frequency AMP and F0 features include mean, standard deviation, DFA, and the coefficients of polynomial curve fitting with degree one and two respectively
Dexterity **Features**	apply the same feature set (includes mean, standard deviation, Q1, Q3, IQR, median, mode, range, skew, kurt, MSE, En, meanTKEO, AR1, DFA) on two groups of tapping intervals: *STAY*: length of time finger stays touching the phone screen *MOVE*: time interval between release of touch to the next touch event
Reaction Features	apply the same feature set on the lags of finger reactions (i.e. the time intervals between the stimulus appearing and the finger touch event), including sum, mean, standard deviation, Q1, Q3, IQR, median, mode, range, skew, kurt, MSE, En, meanTKEO, DFA

[a] Acceleration features are used for both balance and gait tests
[b] TKEO stands for Teager-Kaiser energy operator[111]

polar angle, and azimuth angle). We apply the acceleration features in Table A.2 for these six axes respectively. As the acceleration time series were sampled at irregular time intervals, we applied the Lomb-Scargle periodogram to extract frequency-based features,[114] e.g., the dominant frequency component in Hz and its amplitude. All the acceleration features are used by both the balance and gait tests. To extract the voice features, we first divide the 20-second audio sample into 40 frames leading to 0.5 second frame duration. Then, we tag each frame as containing a "voiced" signal if that frame has amplitude greater than the first quartile of the amplitudes among all frames. Then, for further analysis, we select the longest consecutive run of voiced frames. The length of the largest consecutive run of voiced frames is the "voice duration" feature. Other features extracted from these voiced frames include dominant frequency and amplitude (see Table A.2). Dexterity features are extracted from the *stay* duration, that is, the length of time the finger stays touching the screen, and the *move* duration which is the interval of time between a finger release and the next finger press. The reaction features focus on the lag times of finger reactions (i.e. the time intervals between the stimulus appearing and the finger touch event). The feature extraction library is available at `https://github.com/dashan-emr/mpds`.

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \frac{\lambda_O}{|O|} \sum_{(<\mathbf{x}_i^p, t_i^p>, <\mathbf{x}_j^q, t_j^q>) \in O} L_h(1 - \mathbf{w}^\mathsf{T}(\mathbf{x}_i^p - \mathbf{x}_j^q)) + \frac{\lambda_S}{|S|} \sum_{(<\mathbf{x}_i^p, t_i^p>, <\mathbf{x}_{i+1}^p, t_{i+1}^p>) \in S} \left[\frac{\mathbf{w}^\mathsf{T}(\mathbf{x}_{i+1}^p - \mathbf{x}_i^p)}{t_{i+1}^p - t_i^p} \right]^2$$

Figure B.1: Linear disease severity score objective.

B Disease Severity Score Learning

One general kind of approach for creating a severity score algorithm is based on supervised learning: here, experts evaluate the participants at multiple time points to provide the clinical, "gold-standard" score at each time point (e.g., MDS-UPDRS score). Based on these evaluations, a regression function is estimated that maps features (algorithms such as sensor data variability, complexity and summarized frequency information) derived from the smartphone sensor data collected during the smartphone activities into a continuous or discrete-valued score. The key challenge of using such an approach is that it relies heavily on obtaining a large number of gold-standard clinical evaluations, which are very expensive and time-consuming to collect.

Instead, we used a rank-based machine learning algorithm—*disease severity score learning (DSSL)*[47]—to create the mobile Parkinson disease score (mPDS). In order to estimate a score from feature data, DSSL uses weak supervision[48] where the resulting

labels may have an associated error rate. For example, to estimate mPDS parameters, DSSL exploits example pairs of times that are rank ordered in severity such that the severity of symptoms at time t_i is less than that at time t_j. Using the data collected in this study, such example pairs were easily obtained: for an individual responding to medication, the severity of symptoms at a time right before medication administration is assumed to be higher than that an hour after taking their medications.

Given many such pairs, DSSL estimates a score by optimizing the objective shown in Fig. B.1. Here, x represents a feature vector derived from the sensor data recorded during activities collected using HopkinsPD at a given time. A total of 435 features were computed from the five smartphone-enabled test activities. For example, we computed 126 features from the gait and balance tests each to capture changes in body motion, including the mean, median, standard deviation, range, entropy, and dominant frequency from the tri-axial acceleration time-series. 151 features were computed from the tapping test screen touch events, to quantify attributes such as finger tapping speed (e.g., total number of taps within a given period of time), precision of tapping (e.g., range of tap positions normalized by smartphone screen size), and rhythm and inter-tap interval. Detailed descriptions of the features were previously published elsewhere.[45] Each i, j is a numerical index associated with two distinct timestamps, at times t_i and t_j, at which activities were conducted. Each p, q represents two distinct patient indices. The vector w is a vector of weights estimated by DSSL. To compute the mPDS on a new patient at a given time t given a recording

of their activities at that time and the resulting feature vector x computed from the sensor data collected during these activities, the linear projections $w \cdot x$ are computed. These linear projections are raw and unscaled. To ease interpretability in a clinical setting, the mPDS is scaled between 0 and 100, where values close to 0 reflect low severity while those close to 100 reflect high severity.

The set O is the set of all available pairs of tuples $(< x_i^p, t_i^p >, < x_j^q, t_j^q >)$ that are ordered by severity; from the development cohort, such pairs are computed automatically based on the activities performed at times right before medication administration and those from the hour after. Severity is assumed to be lower post medication administration. In the second term in Fig. B.1, L_h is the Huber loss function. This second term in the objective encourages DSSL to estimate a score that satisfies the severity ordering prescribed by the tuples in set O. We had a total of 3074 such pairs available in the development cohort.

The set S, denoted by pairs of tuples $(< x_i^p, t_i^p >, < x_{i+1}^p, t_{i+1}^p >)$, are obtained based on tests taken at consecutive times within a few hours of each other but without medication administration during the interim period. The third term in Fig. B.1 encourages temporal smoothness for the pairs specified in set S. The coefficients O and S are DSSL regularization parameters and control the relative degree of emphasis on the smoothness between consecutive pairs in the third term of the objective versus maximizing the difference in severity for pairs specified in the second term. These were set using 10-fold cross-validation on the development cohort.

Bibliography

[1] M. S. Donaldson, J. M. Corrigan, L. T. Kohn *et al.*, *To err is human: building a safer health system.* National Academies Press, 2000, vol. 6.

[2] "EHR adoption rates: 20 must-see stats," https://www.practicefusion.com/blog/ehr-adoption-rates/, accessed: 2018-03-24.

[3] B. Rothman, J. C. Leonard, and M. M. Vigoda, "Future of electronic health records: implications for decision support," *Mount Sinai Journal of Medicine: A Journal of Translational and Personalized Medicine*, vol. 79, no. 6, pp. 757–768, 2012.

[4] E. S. Berner, *Clinical decision support systems.* Springer, 2007, vol. 233.

[5] Acute Kidney Injury (AKI) Algorithm. NHS. Accessed April 10, 2018. [Online]. Available: https://www.england.nhs.uk/akiprogramme/aki-algorithm/

[6] Z. Obermeyer and E. J. Emanuel, "Predicting the future—big data, machine

learning, and clinical medicine," *The New England journal of medicine*, vol. 375, no. 13, p. 1216, 2016.

[7] K. E. Henry, D. N. Hager, P. J. Pronovost, and S. Saria, "A targeted real-time early warning score (trewscore) for septic shock," *Science translational medicine*, vol. 7, no. 299, pp. 299ra122–299ra122, 2015.

[8] K. Henry, S. Wongvibulsin, A. Zhan, S. Saria, and D. Hager, "Can septic shock be identified early? evaluating performance of a targeted real-time early warning score (trewscore) for septic shock in a community hospital: Global and subpopulation performance," in *D15. CRITICAL CARE: DO WE HAVE A CRYSTAL BALL? PREDICTING CLINICAL DETERIORATION AND OUTCOME IN CRITICALLY ILL PATIENTS*. American Thoracic Society, 2017, pp. A7016–A7016.

[9] H. Soleimani, K. Henry, A. Zhan, P. Pronovost, N. Rawat, D. Hager, and S. Saria, "Early identification of gastrointestinal bleeding requiring critical care using machine learning," in *D24. CRITICAL CARE: THE OTHER HALF OF THE ICU-UPDATE IN MANAGEMENT OF NON-PULMONARY CRITICAL CARE*. American Thoracic Society, 2017, pp. A7145–A7145.

[10] Y. Liu, K. Gadepalli, M. Norouzi, G. E. Dahl, T. Kohlberger, A. Boyko, S. Venugopalan, A. Timofeev, P. Q. Nelson, G. S. Corrado *et al.*, "Detecting cancer

metastases on gigapixel pathology images," *arXiv preprint arXiv:1703.02442*, 2017.

[11] A. Zhan, S. Mohan, C. Tarolli, R. B. Schneider, J. L. Adams, S. Sharma, M. J. Elson, K. L. Spear, A. M. Glidden, M. A. Little *et al.*, "Using Smartphones and Machine Learning to Quantify Parkinson Disease Severity: The Mobile Parkinson Disease Score," *JAMA neurology*, 2018.

[12] S. Klöppel, C. M. Stonnington, C. Chu, B. Draganski, R. I. Scahill, J. D. Rohrer, N. C. Fox, C. R. Jack Jr, J. Ashburner, and R. S. Frackowiak, "Automatic classification of mr scans in alzheimer's disease," *Brain*, vol. 131, no. 3, pp. 681–689, 2008.

[13] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems*, 2015, pp. 2503–2511.

[14] J. Lin and D. Ryaboy, "Scaling big data mining infrastructure: the twitter experience," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 6–19, 2013.

[15] S. Garber, S. M. Gates, E. B. Keeler, M. E. Vaiana, A. W. Mulcahy, C. Lau, and A. L. Kellermann, "Redirecting innovation in us health care: options to decrease spending and increase value," *Rand health quarterly*, vol. 4, no. 1, 2014.

[16] A. E. Johnson, M. M. Ghassemi, S. Nemati, K. E. Niehaus, D. A. Clifton, and G. D. Clifford, "Machine learning and decision support in critical care," *Proceedings of the IEEE*, vol. 104, no. 2, pp. 444–466, 2016.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[18] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[20] "Terraform," https://www.terraform.io, accessed: 2018-03-23.

[21] "Kubernetes," https://kubernetes.io, accessed: 2018-03-23.

[22] D. Lazer and R. Kennedy, "What we can learn from the epic failure of google flu trends," *Wired. Conde Nast*, vol. 10, 2015.

[23] "Prometheus," https://prometheus.io, accessed: 2018-03-23.

[24] "Grafana," https://grafana.com, accessed: 2018-03-23.

BIBLIOGRAPHY

[25] "Redash," https://redash.io, accessed: 2018-03-23.

[26] "AWS HIPAA Compliance Whitepaper," https://d1.awsstatic.com/whitepapers/compliance/AWS_HIPAA_Compliance_Whitepaper.pdf, accessed: 2018-03-22.

[27] C. M. Torio and R. M. Andrews, "National inpatient hospital costs: the most expensive conditions by payer, 2011: statistical brief# 160," 2006.

[28] M. Singer, C. S. Deutschman, C. W. Seymour, M. Shankar-Hari, D. Annane, M. Bauer, R. Bellomo, G. R. Bernard, J.-D. Chiche, C. M. Coopersmith *et al.*, "The third international consensus definitions for sepsis and septic shock (sepsis-3)," *Jama*, vol. 315, no. 8, pp. 801–810, 2016.

[29] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L.-W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark, "Multiparameter intelligent monitoring in intensive care ii (mimic-ii): a public-access intensive care unit database," *Critical care medicine*, vol. 39, no. 5, p. 952, 2011.

[30] H. Soleimani, J. Hensman, and S. Saria, "Scalable joint models for reliable uncertainty-aware event prediction," *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[31] B. E. Dixon, L. Simonaitis, H. S. Goldberg, M. D. Paterno, M. Schaeffer, T. Hongsermeier, A. Wright, and B. Middleton, "A pilot study of distributed

knowledge management and clinical decision support in the cloud," *Artificial intelligence in medicine*, vol. 59, no. 1, pp. 45–53, 2013.

[32] A. Wright and D. F. Sittig, "Sands: a service-oriented architecture for clinical decision support in a national health information network," *Journal of biomedical informatics*, vol. 41, no. 6, pp. 962–981, 2008.

[33] K. Kawamoto and D. F. Lobach, "Design, implementation, use, and preliminary evaluation of sebastian, a standards-based web service for clinical decision support," in *AMIA Annual Symposium Proceedings*, vol. 2005. American Medical Informatics Association, 2005, p. 380.

[34] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan, "Amazon redshift and the case for simpler data warehouses," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. ACM, 2015, pp. 1917–1923.

[35] K. Sato, "An inside look at google bigquery, white paper," *Google Inc*, 2012.

[36] K. M. Diaz, D. J. Krupka, M. J. Chang, J. Peacock, Y. Ma, J. Goldsmith, J. E. Schwartz, and K. W. Davidson, "Fitbit®: An accurate and reliable device for wireless physical activity tracking," *International journal of cardiology*, vol. 185, pp. 138–140, 2015.

[37] J. Kastrenakes, "Apple watch uses four sensors to detect your pulse," *The Verge*, 2014.

[38] C. G. Goetz, B. C. Tilley, S. R. Shaftman, G. T. Stebbins, S. Fahn, P. Martinez-Martin, W. Poewe, C. Sampaio, M. B. Stern, R. Dodel *et al.*, "Movement disorder society-sponsored revision of the unified parkinson's disease rating scale (mds-updrs): Scale presentation and clinimetric testing results," *Movement disorders*, vol. 23, no. 15, pp. 2129–2170, 2008.

[39] E. R. Dorsey, C. Venuto, V. Venkataraman, D. A. Harris, and K. Kieburtz, "Novel methods and technologies for 21st-century clinical trials: a review," *JAMA neurology*, vol. 72, no. 5, pp. 582–588, 2015.

[40] M. F. Beal, D. Oakes, I. Shoulson, C. Henchcliffe, W. R. Galpern, R. Haas, J. L. Juncos, J. G. Nutt, T. S. Voss, B. Ravina *et al.*, "A randomized clinical trial of high-dosage coenzyme q10 in early parkinson disease: no evidence of benefit," *JAMA neurology*, vol. 71, no. 5, pp. 543–552, 2014.

[41] S. Arora, V. Venkataraman, A. Zhan, S. Donohue, K. Biglan, E. Dorsey, and M. Little, "Detecting and monitoring the symptoms of parkinson's disease using smartphones: A pilot study," *Parkinsonism & related disorders*, vol. 21, no. 6, pp. 650–653, 2015.

[42] A. J. Espay, P. Bonato, F. B. Nahab, W. Maetzler, J. M. Dean, J. Klucken, B. M. Eskofier, A. Merola, F. Horak, A. E. Lang *et al.*, "Technology in parkin-

son's disease: Challenges and opportunities," *Movement Disorders*, vol. 31, no. 9, pp. 1272–1282, 2016.

[43] C. A. Artusi, M. Mishra, P. Latimer, J. A. Vizcarra, L. Lopiano, W. Maetzler, A. Merola, and A. J. Espay, "Integration of technology-based outcome measures in clinical trials of parkinson and other neurodegenerative diseases," *Parkinsonism & related disorders*, vol. 46, pp. S53–S56, 2018.

[44] R. J. Ellis, Y. S. Ng, S. Zhu, D. M. Tan, B. Anderson, G. Schlaug, and Y. Wang, "A validated smartphone-based assessment of gait and gait variability in parkinson's disease," *PLoS one*, vol. 10, no. 10, p. e0141694, 2015.

[45] A. Zhan, M. A. Little, D. A. Harris, S. O. Abiola, E. Dorsey, S. Saria, and A. Terzis, "High frequency remote monitoring of parkinson's disease via smartphone: Platform overview and medication response detection," *arXiv preprint arXiv:1601.00960*, 2016.

[46] "Parkinson's voice initiative," http://www.parkinsonsvoice.org/, accessed: 2017-08-30.

[47] K. Dyagilev and S. Saria, "Learning (predictive) risk scores in the presence of censoring due to interventions," *Machine Learning*, vol. 102, no. 3, pp. 323–348, 2016.

[48] J. Hernández-González, I. Inza, and J. A. Lozano, "Weak supervision and other

non-standard classification problems: a taxonomy," *Pattern Recognition Letters*, vol. 69, pp. 49–55, 2016.

[49] M. R. Lemke, T. Wendorff, B. Mieth, K. Buhl, and M. Linnemann, "Spatiotemporal gait patterns during over ground locomotion in major depression compared with healthy controls," *Journal of psychiatric research*, vol. 34, no. 4, pp. 277–283, 2000.

[50] P. Martínez-Martín, C. Rodríguez-Blázquez, M. Alvarez, T. Arakaki, V. C. Arillo, P. Chaná, W. Fernández, N. Garretto, J. C. Martínez-Castrillo, M. Rodríguez-Violante *et al.*, "Parkinson's disease severity levels and mds-unified parkinson's disease rating scale," *Parkinsonism & related disorders*, vol. 21, no. 1, pp. 50–54, 2015.

[51] A. Antonini, P. Martinez-Martin, R. K. Chaudhuri, M. Merello, R. Hauser, R. Katzenschlager, P. Odin, M. Stacy, F. Stocchi, W. Poewe *et al.*, "Wearing-off scales in parkinson's disease: Critique and recommendations," *Movement Disorders*, vol. 26, no. 12, pp. 2169–2175, 2011.

[52] P. Bailis, K. Olukoton, C. Ré, and M. Zaharia, "Infrastructure for usable machine learning: The stanford dawn project," *arXiv preprint arXiv:1705.07538*, 2017.

[53] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embed-

ding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[54] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.

[55] D. Sculley, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high-interest credit card of technical debt," 2014.

[56] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ml test score: A rubric for ml production readiness and technical debt reduction," *Proceedings of IEEE Big Data*, vol. 2017, 2017.

[57] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," *arXiv preprint arXiv:1711.10160*, 2017.

[58] K. Rong and P. Bailis, "Asap: Automatic smoothing for attention prioritization in streaming time series visualization," *arXiv preprint arXiv:1703.00983*, 2017.

[59] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri, "Macrobase: Prioritizing attention in fast data," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 541–556.

[60] S. Palkar, J. J. Thomas, A. Shanbhag, D. Narayanan, H. Pirk, M. Schwarzkopf, S. Amarasinghe, M. Zaharia, and S. InfoLab, "Weld: A common runtime for

high performance data analytics," in *Conference on Innovative Data Systems Research (CIDR)*, 2017.

[61] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system." in *NSDI*, 2017, pp. 613–627.

[62] J. Hermann and M. Del Balso, "Meet michelangelo: Uber's machine learning platform," *URL https://eng. uber. com/michelangelo*, 2017.

[63] M. W. Jaspers, M. Smeulers, H. Vermeulen, and L. W. Peute, "Effects of clinical decision-support systems on practitioner performance and patient outcomes: a synthesis of high-quality systematic review findings," *Journal of the American Medical Informatics Association*, vol. 18, no. 3, pp. 327–334, 2011.

[64] C. Castaneda, K. Nalley, C. Mannion, P. Bhattacharyya, P. Blake, A. Pecora, A. Goy, and K. S. Suh, "Clinical decision support systems for improving diagnostic accuracy and achieving precision medicine," *Journal of clinical bioinformatics*, vol. 5, no. 1, p. 4, 2015.

[65] T. J. Bright, A. Wong, R. Dhurjati, E. Bristow, L. Bastian, R. R. Coeytaux, G. Samsa, V. Hasselblad, J. W. Williams, M. D. Musty *et al.*, "Effect of clinical decision-support systems: a systematic review," *Annals of internal medicine*, vol. 157, no. 1, pp. 29–43, 2012.

[66] A. K. Jha, C. M. DesRoches, E. G. Campbell, K. Donelan, S. R. Rao, T. G. Ferris, A. Shields, S. Rosenbaum, and D. Blumenthal, "Use of electronic health records in us hospitals," *New England Journal of Medicine*, vol. 360, no. 16, pp. 1628–1638, 2009.

[67] W. C. Willett, J. E. Manson, M. J. Stampfer, G. A. Colditz, B. Rosner, F. E. Speizer, and C. H. Hennekens, "Weight, weight change, and coronary heart disease in women: risk within the'normal'weight range," *Jama*, vol. 273, no. 6, pp. 461–465, 1995.

[68] A. D. Association *et al.*, "Standards of medical care in diabetes—2015 abridged for primary care providers," *Clinical diabetes: a publication of the American Diabetes Association*, vol. 33, no. 2, p. 97, 2015.

[69] B. Christenson, P. Lundbergh, J. Hedlund, and Å. Örtqvist, "Effects of a large-scale intervention with influenza and 23-valent pneumococcal vaccines in adults aged 65 years or older: a prospective study," *The Lancet*, vol. 357, no. 9261, pp. 1008–1011, 2001.

[70] R. Chen, H. Su, M. Khalilia, S. Lin, Y. Peng, T. Davis, D. A. Hirsh, E. Searles, J. Tejedor-Sojo, M. Thompson *et al.*, "Cloud-based predictive modeling system and its application to asthma readmission prediction," in *AMIA Annual Symposium Proceedings*, vol. 2015. American Medical Informatics Association, 2015, p. 406.

[71] T. Desautels, J. Calvert, J. Hoffman, M. Jay, Y. Kerem, L. Shieh, D. Shimabukuro, U. Chettipally, M. D. Feldman, C. Barton *et al.*, "Prediction of sepsis in the intensive care unit with minimal electronic health record data: a machine learning approach," *JMIR medical informatics*, vol. 4, no. 3, 2016.

[72] J. S. Calvert, D. A. Price, U. K. Chettipally, C. W. Barton, M. D. Feldman, J. L. Hoffman, M. Jay, and R. Das, "A computational approach to early sepsis detection," *Computers in biology and medicine*, vol. 74, pp. 69–73, 2016.

[73] E. Choi, A. Schuetz, W. F. Stewart, and J. Sun, "Using recurrent neural network models for early detection of heart failure onset," *Journal of the American Medical Informatics Association*, vol. 24, no. 2, pp. 361–370, 2017. [Online]. Available: http://dx.doi.org/10.1093/jamia/ocw112

[74] S. Arora, V. Venkataraman, S. Donohue, K. M. Biglan, E. R. Dorsey, and M. A. Little, "High accuracy discrimination of parkinson's disease participants from healthy controls using smartphones," in *Movement Disorders*, vol. 28, no. 10, 2013, pp. E11–E11.

[75] W. Dai, J. Doyle, X. Liang, H. Zhang, N. Dong, Y. Li, and E. P. Xing, "Scan: Structure correcting adversarial network for chest x-rays organ segmentation," *arXiv preprint arXiv:1703.08770*, 2017.

[76] B. Jing, P. Xie, and E. Xing, "On the automatic generation of medical imaging reports," *arXiv preprint arXiv:1711.08195*, 2017.

[77] K. Ng, A. Ghoting, S. R. Steinhubl, W. F. Stewart, B. Malin, and J. Sun, "Paramo: a parallel predictive modeling platform for healthcare analytic research using electronic health records," *Journal of biomedical informatics*, vol. 48, pp. 160–170, 2014.

[78] C. J. Bradley, L. Penberthy, K. J. Devers, and D. J. Holden, "Health services research and data linkages: issues, methods, and directions for the future," *Health services research*, vol. 45, no. 5p2, pp. 1468–1488, 2010.

[79] S. N. Murphy, G. Weber, M. Mendis, V. Gainer, H. C. Chueh, S. Churchill, and I. Kohane, "Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2)," *Journal of the American Medical Informatics Association*, vol. 17, no. 2, pp. 124–130, 2010.

[80] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, p. 160035, 2016.

[81] M. Achey, J. L. Aldred, N. Aljehani, B. R. Bloem, K. M. Biglan, P. Chan, E. Cubo, E. Ray Dorsey, C. G. Goetz, M. Guttman *et al.*, "The past, present, and future of telemedicine for parkinson's disease," *Movement Disorders*, vol. 29, no. 7, pp. 871–883, 2014.

[82] M. Little, P. Wicks, T. Vaughan, and A. Pentland, "Quantifying short-term dynamics of parkinson's disease using self-reported symptom data from an internet social network," *Journal of Medical Internet Research*, vol. 15, no. 1, 2013.

[83] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, D. Standaert, M. Akay, J. Dy, M. Welsh, and P. Bonato, "Monitoring motor fluctuations in patients with parkinson's disease using wearable sensors," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 864–873, 2009.

[84] E. Sejdic, K. A. Lowry, J. Bellanca, M. S. Redfern, and J. S. Brach, "A comprehensive assessment of gait accelerometry signals in time, frequency and time-frequency domains," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 3, pp. 603–612, 2014.

[85] M. G. Tsipouras, A. T. Tzallas, D. I. Fotiadis, and S. Konitsiotis, "On automated assessment of levodopa-induced dyskinesia in parkinson's disease," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2011, pp. 2679–2682.

[86] M. Pavel, T. Hayes, I. Tsay, D. Erdogmus, A. Paul, N. Larimer, H. Jimison, and J. Nutt, "Continuous assessment of gait velocity in parkinson's disease from

unobtrusive measurements," in *3rd International IEEE/EMBS Conference on Neural Engineering (CNE'07)*, 2007, pp. 700–703.

[87] T. H. Sanders, A. Devergnas, T. Wichmann, and M. A. Clements, "Remote smartphone monitoring for management of parkinson's disease," in *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '13)*. ACM, 2013, p. 42.

[88] R. K. Begg, M. Palaniswami, and B. Owen, "Support vector machines for automated gait classification," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 5, pp. 828–838, 2005.

[89] R. Levy, W. D. Hutchison, A. M. Lozano, and J. O. Dostrovsky, "High-frequency synchronization of neuronal activity in the subthalamic nucleus of parkinsonian patients with limb tremor," *The Journal of Neuroscience*, vol. 20, no. 20, pp. 7766–7775, 2000.

[90] S. Arora, V. Venkataraman, S. Donohue, K. M. Biglan, E. R. Dorsey, and M. A. Little, "High accuracy discrimination of parkinson's disease participants from healthy controls using smartphones," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 3641–3644.

[91] S. Mazilu, M. Hardegger, Z. Zhu, D. Roggen, G. Troster, M. Plotnik, and J. M. Hausdorff, "Online detection of freezing of gait with smartphones and machine

learning techniques," in *2012 6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, 2012, pp. 123–130.

[92] A. Weiss, S. Sharifi, M. Plotnik, J. P. van Vugt, N. Giladi, and J. M. Hausdorff, "Toward automated, at-home assessment of mobility among patients with parkinson disease, using a body-worn accelerometer," *Neurorehabilitation and Neural Repair*, vol. 25, no. 9, pp. 810–818, 2011.

[93] A. Tsanas, M. A. Little, P. E. McSharry, J. Spielman, and L. O. Ramig, "Novel speech signal processing algorithms for high-accuracy classification of parkinson's disease," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 5, pp. 1264–1271, 2012.

[94] R. LeMoyne, T. Mastroianni, M. Cozza, C. Coroian, and W. Grundfest, "Implementation of an iphone for characterizing parkinson's disease tremor through a wireless accelerometer application," in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, vol. 56, 2010.

[95] H. Hodson, "Apple researchkit and watch will boost health research," *New Scientist. https://www. newscientist. com/article/dn27123-apple-researchkit-and-watch-will-boost-health-research/. Accessed*, pp. 08–21, 2015.

[96] B. M. Bot, C. Suver, E. C. Neto, M. Kellen, A. Klein, C. Bare, M. Doerr, A. Pratap, J. Wilbanks, E. R. Dorsey *et al.*, "The mpower study, parkinson dis-

ease mobile data collected using researchkit," *Scientific data*, vol. 3, p. 160011, 2016.

[97] Y.-F. Y. Chan, P. Wang, L. Rogers, N. Tignor, M. Zweig, S. G. Hershman, N. Genes, E. R. Scott, E. Krock, M. Badgeley *et al.*, "The asthma mobile health study, a large-scale clinical observational study using researchkit," *Nature biotechnology*, vol. 35, no. 4, p. 354, 2017.

[98] B. Suffoletto, P. Gharani, T. Chung, and H. Karimi, "Using phone sensors and an artificial neural network to detect gait changes during drinking episodes in the natural environment," *Gait & posture*, vol. 60, pp. 116–121, 2018.

[99] D. E. Webster, C. Suver, M. Doerr, E. Mounts, L. Domenico, T. Petrie, S. A. Leachman, A. D. Trister, and B. M. Bot, "The mole mapper study, mobile phone skin imaging and melanoma risk data collected using researchkit," *Scientific data*, vol. 4, p. 170005, 2017.

[100] B. Byrom, L. Simpson, J. Lee, E. Flood, C. Cassedy, A. Gadala-Maria, W. Muehlhausen, M. Mc Carthy, and B. Skerritt, "Applicability of apple research kit to deliver cognitive testing in clinical trials: Results of a pilot study," in *ISPOR 22nd Annual International Meeting*, 2017.

[101] N. Kostikis, D. Hristu-Varsakelis, M. Arnaoutoglou, and C. Kotsavasiloglou, "A smartphone-based tool for assessing parkinsonian hand tremor," *IEEE journal of biomedical and health informatics*, vol. 19, no. 6, pp. 1835–1842, 2015.

[102] R. LeMoyne and T. Mastroianni, "Use of smartphones and portable media devices for quantifying human movement characteristics of gait, tendon reflex response, and parkinson's disease hand tremor," in *Mobile Health Technologies*. Springer, 2015, pp. 335–358.

[103] H. Kim, H. J. Lee, W. Lee, S. Kwon, S. K. Kim, H. S. Jeon, H. Park, C. W. Shin, W. J. Yi, B. S. Jeon *et al.*, "Unconstrained detection of freezing of gait in parkinson's disease patients using smartphone," in *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*. IEEE, 2015, pp. 3751–3754.

[104] D. Harris, S. Abiola, K. Biglan, E. Dorsey, M. Little, S. Saria, and A. Zhan, "Smartphone-pd: Preliminary results of an mhealth application to track and quantify characteristics of parkinson's disease in real-time," *Movement Disorders*, vol. 30, pp. S273–S274, 2015.

[105] S. O. Abiola, K. M. Biglan, E. R. Dorsey, D. A. Harris, M. A. Little, S. Saria, and A. Zhan, "Smartphone-pd: Preliminary results of an mhealth application to track and quantify characteristics of parkinson disease in real-time," *Technology*, vol. 18, p. 8, 2015.

[106] A. Zhan, S. Mohan, M. Elson, E. Dorsey, A. Terzis, and S. Saria, "Patient clustering through a mobile-based PD Severity Score," in *MOVEMENT DIS-*

ORDERS, vol. 32. WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, 2017.

[107] A. Zhan, S. Mohan, M. Elson, E. Dorsey, A. Terzis, and S. Saria, "Efficacy of smartphone-enabled active tests in automated PD severity assessment," in *MOVEMENT DISORDERS*, vol. 32. WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, 2017.

[108] A. Zhan, S. Mohan, M. Elson, E. Dorsey, A. Terzis, and S. Saria, "PDSS: a novel mobile-based Parkinson's disease severity score," in *MOVEMENT DISORDERS*, vol. 32. WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, 2017.

[109] R. Schneider, J. Adams, M. Elson, C. Tarolli, S. Sharma, A. Glidden, T. Felong, A. Zhan, R. Korn, S. Goldenthal *et al.*, "Feasibility of using a smartphone application for the objective evaluation of parkinson's disease," in *MOVEMENT DISORDERS*, vol. 32. WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, 2017.

[110] M. Little, P. McSharry, I. Moroz, and S. Roberts, "Nonlinear, biophysically-informed speech pathology detection," in *Proceedings of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006).*, vol. 2, pp. II–1080–II–1083.

[111] J. F. Kaiser, "On a simple algorithm to calculate the 'energy' of a signal,"

in *1988 International Conference on Acoustics, Speech, and Signal Processing (ICASSP-88).*, pp. 381–384.

[112] D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso, "Preprocessing techniques for context recognition from accelerometer data," *Personal and Ubiquitous Computing*, vol. 14, no. 7, pp. 645–662, 2010.

[113] S. Arora, V. Venkataraman, A. Zhan, S. Donohue, K. Biglan, E. Dorsey, and M. Little, "Detecting and monitoring the symptoms of parkinson's disease using smartphones: A pilot study," *Parkinsonism & Related Disorders*, 2015.

[114] J. D. Scargle, "Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data," *The Astrophysical Journal*, vol. 263, pp. 835–853, 1982.

ANDONG ZHAN

Address: 3400 North Charles Street
Dept. of Computer Science, Malone Hall
Johns Hopkins University
Baltimore, MD 21218, USA

Email: zad522@gmail.com, azhan2@jhu.edu
My Publications: Google Scholar
My Links: Homepage, LinkedIn, My Tech Blog, Stack Overflow

ABOUT ME

I'm a Ph.D. graduate in Computer Science whose mission is to build novel computer systems and machine learning algorithms to solve real-world problems in healthcare. I began my research at Johns Hopkins University and I have designed and deployed two game-changer systems for healthcare which directly influence thousands of inpatients and outpatients: For example, we first deployed AI into clinical decision support in Johns Hopkins Hospitals since November 2017 and generated hundreds of real-time alerts for those who may develop sepsis; Our smartphone-based Parkinson severity score mPDS has been created by monitoring thousands of patients remotely and its machine learning-based score is highly correlated with standard Parkinson assessments. This work has been reported by Nature Outlook, JHU Hub, etc. Thus far, I have published more than 20 peer-reviewed articles in the areas of computer science and clinical research, including IEEE Communications, ACM SenSys, JAMA Neurology, Parkinsonism & Related Disorders, and Critical Care Medicine. My work has been cited in excess of 200 times by researchers from all over the world. I'll be officially graduate in May 2018 and I am looking for my next adventure.

EDUCATION

Ph.D, Computer Science — September 2010 - May 2018
John Hopkins University, USA
Main Courses: Network Embedded Systems, Parallel Programming, and Machine Learning.

Master of Engineering, Computer Software and Theory — September 2007 - May 2010
Nanjing University, Nanjing, China
Main Courses: Wireless Networks, Data Mining, Algorithm, Design Patterns.

Bachelor of Science, Computer Science and Technology — September 2003 - May 2007
Nanjing University, Nanjing, China
Main Courses: Data Communication and Network, Data Structures, Operating Systems, Digital Image Processing, Database, Software Engineering, and Principles of Compiling.

SELECTED RESEARCH EXPERIENCE

Research on Computer Systems for Healthcare
Advised by Prof. Andreas Terzis and Prof. Suchi Saria at JHU. — September 2010 - present

- **Smartphone-based Parkinson disease severity monitoring.** I built HopkinsPD, an Android application released at 2014, which collected smartphone activities from nearly one thousand of Parkinson patients and controls over the last three years. To transform smartphone measures to a clinically verified severity score, I developed mPDS – the mobile Parkinson Disease Score – using a novel machine learning technology, called Disease Severity Score Learning, which can generate severity score from clinical comparison. The mPDS achieved high correlation with conventional clinical Parkinson severity assessment and meanwhile, it can be performed by the patients anytime anywhere. My work has been

accepted by JAMA Neurology in 2018 and reported by Nature magazine 2016.

- **CDS-Stack: Deliver Online Clinical Decision Support System using EHR and Machine Learning.** Over the last decade, American hospitals have adopted electronic health records (EHRs) widely. In the next decade, incorporating EHRs with clinical decision support (CDS) together into the process of medicine has the potential to change the way medicine has been practiced and advance the quality of patient care. It is a unique opportunity for machine learning (ML), with its ability to process massive datasets beyond the scope of human capability, to provide new clinical insights that aid physicians in planning and delivering care, ultimately leading to better outcomes, lower costs of care, and increased patient satisfaction. However, applying ML-based CDS has to face steep system and application challenges. No unified platform is there to support ML and domain experts to develop, deploy, and monitor ML-based CDS. Build ML-based CDS from scratch can be expensive and time-consuming. And no end-to-end solution is available for machine learning algorithms to consume heterogenous EHRs and deliver CDS in real-time.

  In this project, CDS-Stack, an open cloud-based platform, is introduced to help researchers and developers to deploy ML-based CDS into healthcare practice. The CDS-Stack integrates various components into infrastructure for the development, deployment, and monitoring of the ML-based CDS. It provides an ETL engine to transform heterogenous EHRs, either historical or online, into a common data model (CDM) in parallel so that ML algorithms can directly consume health data for training or prediction. It introduces both pull and push-based online CDS pipelines to deliver CDS in real-time. The CDS-Stack has been applied and deployed at Johns Hopkins Medical Institute (JHMI) to deliver the TREWS sepsis alert since Nov 2017 and begins to show promising results.

- **Accurate Caloric Expenditure of Bicyclists using Cellphones**: Biking is one of the most efficient and environmentally friendly ways to control weight and commute. To precisely estimate caloric expenditure, bikers have to install a bike computer or use a smartphone connected to additional sensors such as heart rate monitors worn on their chest, or cadence sensors mounted on their bikes. However, these peripherals are still expensive and inconvenient for daily use. This work poses the following question: *is it possible to use just a smartphone to reliably estimate cycling activity?* We answer this question positively through a pocket sensing approach that can reliably measure cadence using the phone's on-board accelerometer with less than 2% error. Our method estimates caloric expenditure through a model that takes as inputs GPS traces, the USGS elevation service, and the detailed road database from OpenStreetMap. The overall caloric estimation error is 60% smaller than other smartphone-based approaches. Finally, the smartphone can aggressively duty-cycle its GPS receiver, reducing energy consumption by 57%, without any degradation in the accuracy of caloric expenditure estimates. This is possible because we can recover the bike's route, even with fewer GPS location samples, using map information from the USGS and OpenStreetMap databases. This work has been published at ACM SenSys 2012.

TECHNICAL EXPERIENCE

The most familiar

- Programming skills: Python (asyncio, pandas, sklean, jupyter notebook, falcon), PostgreSQL, Android
- Machine learning algorithms: random forests, linear regression, logistic regression, SVM, K-means, GMM, and Bayesian network.
- Cloud-based infrastructure and platform services: AWS, Kubenetes, terrafrom
- Network protocols: TCP/IP, UDP, HTTP(S), REST, SOAP, JSON
- Software tools: Linux shell, Github, redash, grafana, prometheus, Android Studio
- Epic EHR certificates:
  - Clarity Data Model – EpicCare Inpatient Certification

- Research Informatics Innovator – Core Certification
- Chronicles Database Programmer Certification
- Interconnect Programmer Certification

Open source code

- CDS-Stack: An open cloud-based platform for developing machine learning based clinical decision support system.
- HopkinsPD: An Android application to collect Parkinson disease related activities.
- mPDS: An smartphone-based Parkinson disease severity score and feature extraction code.

PROFESSIONAL EXPERIENCE

Editor at Elsevier DIB (Data In Brief) Journal March 2017 - now
Computer Science domain

Adjunct member at Center for Clinical Data Analysis (CCDA), Johns Hopkins Medicine
June 2016 - now

- Extracted (Electronic Health Records) EHR from Johns Hopkins Medicine (Epic Clarity database) and built EHR database for data analysis.

Software Engineering Internship at Qualcomm
Qualcomm Research Berkeley, Berkeley, CA June 2014 - August 2014

- Built a demonstration system on Android tablets for dynamic adaptive streaming technologies and application layer FEC technologies.

Software Engineering Internship at Hillcrest Labs
Advanced R&D Team in Hillcrest Labs, Rockville, MD June 2013 - August 2013

- Designed and evaluated motion prediction algorithms to reduce delay of remote controller for smart TVs.

Teaching Assistant for Course: Computer Networks, JHU Spring 2013

- Guided and graded homework and projects, including 1) a simplified Dropbox system with server and Android client app; 2) TinyControl: a simplified congestion control protocol; 3) a simple router implementation on Mininet.

SELECTED HONORS

2017 MDS Congress Student Travel Grant
2012 ACM SenSys Student Travel Grant
2010/2011 fellowship of Computer Science Graduate Students at JHU, 2010
The Price of Excellent Master Thesis at Computer Science Department of NJU, 2010
IBM Chinese Excellent Student Scholarship, 2009
HONORED MENTION in ACM Mobicom 2009 Demos and Exhibits